e-IDea project (IWT Tetra 070140)
Final Report:
Developing secure applications using the Belgian eID technology
Version 1.0

Vincent Naessens (KaHo Sint-Lieven vakgroep IT, MSEC)
Jorn Lapon (KaHo Sint-Lieven, Research Unit IT, MSEC)
Bram Verdegem (KaHo Sint-Lieven, Research Unit IT, MSEC)
Bart De Decker (KULeuven Dept. of Computer Science, SecAnon)
Pieter Verhaeghe (KULeuven Dept. of Computer Science, SecAnon)

User group:
4all networks, Alphatronics, Arena IT Solutions, Atos Worldline, Commeto, Custodix, Deloitte, DSoft, DSP Valley, dzine, Fedict, IBM, Intesi, KMO-IT, lin-k, Profon, Roadbyte, Zetes, ZION Security

October 21, 2009
`http://www.msec.be/eidea`

Contents

Chapter 1

Introduction

Belgium introduced an electronic identity card as one of the first countries in Europe in 2002. The card allows Belgian citizens to identify, to authenticate and to sign electronic documents. The roll-out of the eID card will be completed by the end of 2009. This means that every Belgian citizen will have an eID card at that time. This is an important milestone for all stakeholders that are involved in the Belgian eID technology: the federal and local governments, the developers of the eID card, the developers and distributors of infrastructure and hardware (i.e. middleware, revocation lists, card readers ...), the application developers - SMEs as well as large companies and the end users of the Belgian eID technology.
The finalization of the roll-out may lead to a boom of new applications in multiple domains. Moreover, many other countries may rely on the expertise in the domain of eID technology that is currently available in Belgian companies. Therefore, the quality of eID applications is crucial. The chip has a tamper-proof resistant memory and uses well-known cryptographic blocks (i.e. asymmetric key technology). However, developing secure eID applications is not trivial. One secure component (i.e. the eID chip) does not necessarily result in a secure eID application. Moreover, many other requirements need to be tackled in advanced eID applications. Usability is essential. Privacy is also an important concern in many eID applications (especially when the eID card is used in the commercial domain). Moreover, eID applications must comply with national and European legislation.
This report summarizes the results of the eIDea project (i.e. a technology transfer project funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders). The consortium consists of two research groups (i.e. MSec - Research Unit IT at Katholieke Hogeschool Sint-Lieven and DistriNet - the Department of Computer Science at KULeuven) and multiple SMEs and large companies in Flanders. The project aims at offering software support for future Belgian eID applications. The outline of the report is summarized below.
Chapter 2 gives an overview of the Belgian eID technology. The components of the eID card and the software that is currently offered by the Belgian government are described. Next, we focus on the security and privacy properties of the current eID card. It will be clear that applications (especially in the commercial domain) may exploit those weaknesses. Finally, eID initiatives in other countries are listed.
Chapter 3 mainly focuses on cryptographic building blocks for secure eID applications. Such applications typically use the eID card to retrieve other tokens (such as pseudonym certificates or anonymous credentials). The latter are used to authenticate to service providers. More-

over, a framework is presented that can integrate advanced cryptographic building blocks. Though, it offers simple and inituitive APIs for authentication and digital signatures using alternative cryptographic protocols. Hence, the complexity of the cryptographic operations is hidden from application developers.
Chapter 4 presents several eID applications that are built on top of the framework (i.e. home automation systems, eTicketing applications and ePolls). In the first applicatoin, the eID card is used to access and modify settings in the home automation system. In the other applications, individuals register with the eID card (i.e. the card is used in a bootstrap procedure). After successful registration, the users can order tickets or express their opion using a privacy-friendly token. Multiple alternatives are discussed (i.e. using pseudonym certificates and anonymous credentials). Using the eID card as a bootstrap has multiple advantages compared to direct use of the eID card. It is shown that this approach may lead to more privacy-friendly applications. However, the complexity of the design and implementation increases. The framework may offer support to accelerate the development of more advanced eID applications.
Chapter 5 presents multiple eID extensions that can be used in a broad range of application domains. First, an authentication protocol is discussed. The protocol tackles some privacy and security weaknesses that arise when using the eID for client authentication in SSL. Second, eID proxy certificates are presented. They increase the functionality and mobility of eID applications. However, to comply with the privacy legislation, Belgian eID proxy certificates need to slightly differ from the standards for proxy certificates. A third extension shows how the eID card can be used to generate symmetric keys. The keys can be used to store sensitive information (such as passwords, tickets, certificates, ...) securely at a remote location. Finally, the feasibility of the eID extensions is validated through the design of several applications in different domains.
Chapter 6 points to future eID technologies. Smart card solutions as well as solutions on mobile devices are discussed. This chapter ends with general conclusions.

Chapter 2

The Belgian eID Technology

The Belgian Electronic Identity Card was introduced in 2002. The card enables Belgian citizens to prove their identity digitally and to sign electronic documents. Today, many application developers integrate e-ID plugins in their applications. Users may even be forced to use their e-ID card to access certain services. However, inappropriate use of the card may cause harm to the card holder.
This chapter starts with a general overview of the Belgian eID card, followed by a privacy and security assessment of the Belgian ID technology. Finally, a short description and pointers to eID technologies developed in other countries are listed.

2.1 Overview

2.1.1 Contents of the Belgian eID card

Private information such as the owner's name, birthdate and -place, address, digital picture and National Registration Number is stored in three separate files: an identity file, an address file and a picture file. The files are signed by the National Registration Bureau (NRB). The National Registration Number (NRN)[1] is a unique nation-wide identification number that is assigned to each natural person.

Two key pairs are stored on the eID card. One key pair is used for authentication, the other is used is for signing. The (qualified) e-signatures are legally binding. The public keys are embedded in a certificate which also contains the NRN and the name of the card holder. The private keys are stored in a tamper-proof part of the chip and can only be activated (not *read*) with a PIN code. Authentication is single sign-on, i.e. the PIN code is only required for the first authentication. For signing, a PIN code is needed for each signature[57].

2.1.2 Official middleware

The cryptographic functionalities in the Belgian eID card are accessed through middleware [20]. Applications typically interact with the card via a simple API [51] offered by this middleware. If a document needs to be signed, the middleware passes a hash of the document to the card. Similarly, a hash of the challenge is passed to the card for authentication purposes.

[1]The NRN has a fixed format: it consists of a 13 digit number structured as yymmdd-nnn.cc where yy/mm/dd is th birthdate, nnn is a 3 digit number (which is odd for males and even for females) and cc 2 control digits (mod 97; with a twist for people born after 2000)

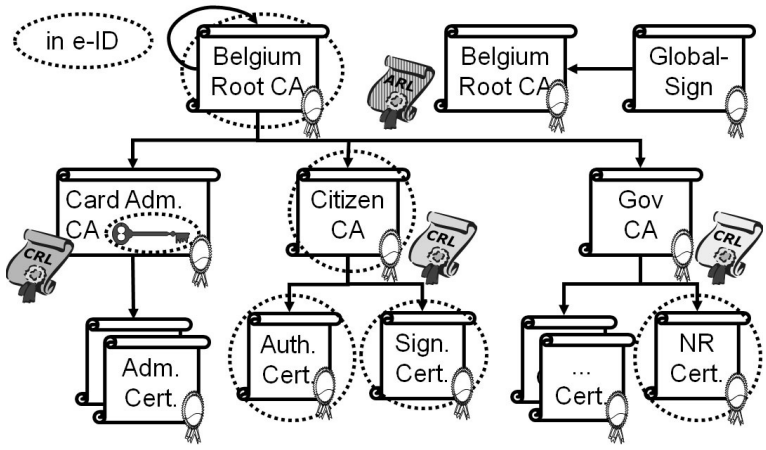


Figure 2.1: Belgian Public Key Infrastructure

When an application wants to authenticate or sign a document with the eID card, the middleware requests the user to enter his PIN code in order to activate the appropriate private key. The middleware can also verify the validity of the certificates (using CRL or OCSP). It is important to note that the use of the official middleware is not mandatory. Several alternatives, developed by different companies, are available.

2.1.3 Belgian Public Key Infrastructure

The certificates on the eID card are part of a larger hierarchical infrastructure, the Belgian Public Key Infrastructure (be-PKI) [49]. The hierarchy is illustrated in figure 2.1. The citizen's signature and authentication certificates are issued by a *Citizen_CA* which is certified by the *Belgium_Root_CA*. Other governmental CAs such as the *Card_Admin_CA* and *Government_CA* also have certificates issued by the *Belgium_Root_CA*. The former can update the eID card. The latter certifies the National Registration Office (NRO) which signs the identity and address files and offers other services in the public sector. The *Belgium_Root_CA* has two certificates. The first is a self-signed certificate, that allows for offline validation of the signature and authentication certificates on the eID card. The second certificate is issued by GlobalSign. The latter is typically known to popular applications (such as browsers) and allows for the automatic validation of electronic signatures. The PKI provides Authority Revocation Lists (ARL) and Certificate Revocation Lists (CRL) [3] that keeps the serial numbers of the revoked certificates.
In the figure, the items (certificates or public keys) which are encircled with a dotted line, are also stored on the eID card.

2.1.4 Applications

The Belgian eID card opens up new perspectives for the government, its citizens, service providers and application developers. The government aims at increasing the quality of services that are offered to its citizens by implementing e-government applications [12]. For instance, using the eID card, individuals can consult personal information stored in govern-

mental databases [4, 9]. They can also request official documents [11] or submit and consult tax declarations [16]. Players in both the public sector and the profit sector also benefit from the deployment of the eID. New applications are developed and existing applications are extended with plugins that support eID authentication and eID signing [14, 10]. Some examples are e-Access applications [15, 30], e-Banking [8], e-Commerce [17], e- Health [2], ... The Flemish government also funds research projects that focus on eID integration [1, 35].

2.2 Security and privacy assesment eID technology

Careless use of the e-ID card and middleware may cause economic and psychosocial damage. This section focuses on security and privacy risks related to the use of the Belgian Electronic Identity card [38, 31]. First, we discuss the security and privacy threats assuming that both the middleware and the application are trustworthy. Next, we discuss attacks that can be performed by malicious applications. Thereafter, the assumption of trusted middleware is omitted: we show how the middleware can easily be disabled or modified. Finally, we discuss how implementation flaws can lead to privacy/security incidents.

2.2.1 Trustworthy middleware and application

When both middleware and application software are trustworthy, security and privacy risks still exist. Note that these threats apply to all card holders using their eID card.

Privacy risks. During an authentication to a service provider using the e-ID card, the authentication certificate is automatically sent to the service provider. Since the certificate contains the unique NRN, all actions performed by the same citizen can be linked. The date of birth and gender of the individual can also be derived from the NRN (see footnote [1]). Similarly, all signatures created with the same e-ID card can easily be linked.

Moreover, once the citizen entered his PIN code to authenticate to a certain website, the private authentication key remains activated as long as the card is not removed from the card reader. Hence, the PIN code is no longer required for subsequent authentications to other sites. When the user then browses to multiple sites that require e-ID authentication, authentication is performed transparently. This implies that users are unaware that identity information (i.e. the authentication certificate) is transferred to these sites. Another threat is related to the default settings of the middleware. When inserting the e-ID card in the card reader, the authentication and signing certificates are stored in persistent memory of the PC by default. Disabling this option prevents that these certificates are stored automatically. However, it also prevents users to authenticate with the e-ID card in certain applications (e.g. Internet Explorer).

Security risks. Documents that are signed with the signature key K_{Sig} on the Belgian e-ID card are legally binding. Several applications offer support to sign digital documents with the e-ID card. When a user wants to sign a document in Open Office or Adobe Reader, he can choose to sign the document with K_{Aut} or K_{Sig}. An unaware or malicious user may select his *authentication key* K_{Aut} to sign a document. Hence, verifiers must check that a valid *signing key* was used.

		Privacy threats	Security threats
Trustworthy middleware	**Trustworthy application**	• Sensitive certificate attributes (NRN) • Unaware authentication • Automatic certificate retrieval & storage on local disk	• Signatures with incorrect key • PIN code capturing
	Compromised application	• Distributing picture, identity and address • Collecting personal records	• Surreptitious authentication • Surreptitious signatures
Compromised middleware		• Identity theft	• Denial-of-service
Implementation flaws		• Insecure communication channels • Inappropriate privacy policies	• Insecure communication channels • Incomplete validity checks • Problematic (unverifiable) certificate chains

Figure 2.2: Overview of security and privacy threats

2.2.2 Trustworthy middleware and compromised application

Compromised applications can circumvent the security and privacy measures in the middleware. To prevent malicious software to access the e-ID card directly, a privacy service which locked the card reader was initially developed. However, when this service is stopped, malicious programs are no longer prohibited to access the card directly. Sometimes, the user has to disable the service [7] to allow certain applications to read other (non-Belgian e-ID) smart cards. In the current version of the middleware, the privacy service is no longer available.

Privacy risks. The picture, identity and address files on the e-ID card are not PIN-protected. In normal circumstances, the middleware will request the user's consent to access these files. However, if the name of a program that makes use of the middleware API is the same as the name of a trusted application (such as beidgui.exe or beidsystemtray.exe), the user's consent is no longer required. Moreover, a program connecting directly to the card reader can collect these files and distribute them over the Internet. This is especially problematic when children use their e-ID card to login at a "secure" chat box, or if the card is used to get access to a building (e.g. sauna complex) since the identity and address data can easily be misused. Not only personal information that is stored on the e-ID card can be collected. After a user has entered his PIN code once, a compromised application can transparently authenticate to multiple websites (such as tax-on-web [16], NRN dossier [11], be-health which keeps medical records [2]) and collect even more identifying data or even modify this data. The application can then forward this information for later misuse (e.g. identity theft).

Security risks. A malicious application can deceive the user and let the e-ID card sign a different document than the one intended. Assume that a user wants to sign a document. The application can send a different document to the middleware and transparently forward the signature to a malicious host. Moreover, since the PIN code for authentication only has to be entered once, an application can connect to other e-ID protected sites and order goods in the name of the cardholder. The PIN code, captured with for instance a key logger can be exploited for clandestine authentication or signing.

2.2.3 Compromised middleware

A malicious program running with administrative privileges can also compromise the middleware. When the modified middleware has a similar GUI, the user will probably not notice any difference. Compromised middleware can have full control to identity information on the card and the PIN code can be intercepted and stored unless a card reader with separate keypad is used. Thereafter, it can be used for fetching more information from protected websites during a longer period. Malicious middleware can have full control over the authentication and signing functions and can even imperceptably change the PIN code. Hence, the card can no longer be used by the owner (denial of service).

The same attacks are possible when an individual uses his card on an untrusted platform. Assume that a malicious insurance company asks an individual to insert his e-ID card and enter his PIN code on a company's workstation to sign the insurance contract. Meanwhile, if no card reader with seperate keypad is used, the PIN code can be intercepted and personal records can be retrieved by that workstation, by transparently authenticating to trusted web

services that keep records of that user. The insurance company can then stipulate other conditions in the contract based on the collected info.

2.2.4 Implementation flaws

Many application developers include e-ID plugins in their software. However, not all developers are security specialists. This may lead to implementation flaws that can result in privacy and security problems. For instance, web servers may force users to authenticate over an insecure connection. If so, all identity information that is transferred can be eavesdropped. Moreover, web servers do not always check the validity and revocation status of the authentication/signing certificate. Many web servers also have credential management problems (even servers that are administered by the government such as [11]). When a client connects to a secure site, the server should return a chain of certificates required to verify the server certificate. Many servers are not configured appropriately. Hence, the user receives a warning that the server certificate could not be verified. Many users ignore that warning and proceed. However, connecting to a malicious web server will often result in the same warning.

2.3 Other eID solutions

Besides Belgium, several other countries have introduced a similar electronic identity. This section gives a short description of these solutions and refers to interesting information. We discuss both card specific and card independent solutions. Card specific solutions are bound to one type of card, provided by the government. Card independent solutions are not bound to one type of card, but can be included in other cards such as SIM cards and credit cards.

2.3.1 Card specific eID solutions

European Passport

- EU Passport Specification: This document describes solutions for chip enabled European passports.
  url: `http://ec.europa.eu/justice_home/doc_centre/freetravel/documents/doc/c_2006_2909_en.pdf`

- Regulating a European eID: A preliminary study on a regulatory framework for entity authentication and a pan-European eID.
  url: `http://www.fidis.net/resources/deliverables/hightechid/int-d36000/doc/25/`

Dutch DigiD and eNIK

DigiD is a government-wide authentication service and consists of multiple levels of security. The basic level consists of a username and password. For present services, this was found satisfactory. The middle level consists of sms-messages during authentication. The high level solution consists of an electronic identity card (eNIK), however, its development is still in an early stage.

- Digid, a digital identity for dutch citizens url: `http://www.digid.nl/`.

Portuguese Cartao comum do cidadão

The Portuguese electronic eID is a smart card that provides visual identity authentication with biometrics (photo and fingerprint) and allows for electronic signatures.

French Vitale 2

The Vitale 2 card is the future French e-health card. It complies with the identification, authentication and signature standards (ISA). It has a built-in crypto-processor for public key operations. Moreover it contains a picture of the cardholder. Full roll-out of the cards is planned for 2010.

German ePass

The German electronic Passport contains a microchip holding the personal data, such as name, surname, date of birth, nationality and the holder's picture. The second generation contains 2 digitally stored fingerprints.

- Official portal on the German e-Health card
  url: `http://www.gesundheitskarte-sh.de/`

Italian CIE

The Italian eID card (CIE) contains a microchip, a machine readable zone and optical memory. It keeps a set of personal data, including the holder's fiscal code, blood group, and fingerprint scans. A PIN code is required to retrieve the personal data, biometric key and digital signature stored on the card. The cardholder's fingerprint-template is stored in both the microchip and the optical memory and does not allow fingerprint reconstruction.

- Support and information about the Italian electronic identity card (CIE)
  url: `http://www.ancitel.it/cie/`

Spanish DNI electronico

The Spanish eID card (DNI Electronico) enables the card holder to digitally sign electronic documents and contracts, to identify himself and to authenticate to (remote) service providers. These functionalities are protected with a PIN code.

- Official portal for the Spanish eID card (DNI)
  url: `http://www.dnielectronico.es/`

Swedish eID card

The Swedish eID card (nationellt identitetskort) is not compulsory and does not replace previous paper ID cards. It contains two chips. One of them is contactless, containing identity information (digital picture). The other one is a traditional contact chip for strong authentication and signing.

- `http://www.police.se/inter/nodeid=3929&pageversion=1.jsp`

Estonian ID card

The Estonian ID card allows identification, secure authentication and the creation of legally binding digital signatures. The chip contains personal data as well as certificates for authentication and signing. Their associated private keys are protected with PIN codes. The certificates contain only the holder's name and a personal code (i.e. the national ID code).

- `http://www.pass.ee`

2.3.2 Card independent eID solutions

Austrian Bürgerkarte

The Austrian Citizen card (Bürgerkarte) is a card independent solution based on electronic signatures and strong authentication. It enables citizens to securely access public services and to complete administrative procedures electronically.
The Austrian eID concept is not restricted to one single type of citizen card. Any card that allows digital signing and storing personal data can be used as a Citizen Card. Membership cards, bank cards and even mobile technologies can implement the Bürgerkarte.

- Official portal on the Austrian Bürgerkarte
  url: `http://www.buergerkarte.at/`

Finish FINEID

Finland first implemented a citizen card (FINEID) containing certificates for digital signing, strong authentication and encryption. Later on, FINEID evolved to a card independent solution based on certificates (i.e. the citizen certificate) that can be embedded in different types of cards such as Visa and SIM cards. The certificate contains, among other information, the citizens first name, its last name and an electronic client identifier. The Population Register Centre also issues Organisation Certificates to companies and legal persons.

- Official portal on the finish FINEID card
  url: `http://www.fineid.fi/`
- *Document* : Guidelines for Developing Applications using FINEID card

2.4 Conclusion

The Belgian eID can replace current identification and authentication mechanisms in existing applications. Moreover, digital documents can be signed electronically. However, introducing the Belgian eID card in commercial applications is no sinecure. Moreover, strong authentication is often not necessary in many applications. Inappropriate use of the electronic eID card introduces a number of security and privacy risks. Irresponsible use may lead to the divulgement of important personal data resulting in economic and/or psychosocial damage. However, many citizens are unaware of these pitfalls. On the contrary, unexperienced users are encouraged (or even obliged) to use the e-ID card for many purposes.
Across Europe, different eID solutions are emerging. The motivation for their deployment varies from country to country, and hence also their ability to interoperate. In a highly interconnected world, interoperability is a key concern for application developers. Although there

are several initiatives to accelerate interoperability across national eID schemes, it is by far not trivial to realise full interoperability (not only from the technical viewpoint but also from the legal perspective).

Chapter 3

Building blocks for secure eID applications

A good knowledge of basic building blocks is required to build secure and privacy-friendly eID applications. This chapter gives an overview of existing mobile technologies and cryptographic building blocks. Also, an identity framework is introduced that supports advanced cryptographic technologies while hiding their complexity.

3.1 Mobile technologies

3.1.1 Card technologies

Smart Cards

Description of technology.
Smart cards are plastic cards with an embedded microprocessor. The microprocessor allows for a secure tamper-resistant environment in which access to private information can be restricted. We discuss both contact and contactless cards.

Evaluation of technology.

Security

- Pro:
  - *Tamper-resistance.* Smart cards are designed to be secure tamper-resistant devices.
  - *Compatibility.* Since the start of the development of smart cards, compatibility with existing standards is important. Moreover, a common standard was developed to enable smart card readers to read any type of smart card.
  - *Reliability.* With respect to magnetic stripe cards, smart cards are more reliable.
  - *Data storage.* Confidential data such as PIN codes and secure keys can be securely stored on the card. It can store up to 64kB and newer types can store even more, while magnetic cards can only hold about 1000 bits.
  - *Offline transactions.* Smart cards can store and process confidential information making offline transactions (e.g. signing and authentication) possible.

- Contra:
  - *Physical attacks.* Physical attacks on the embedded integrated circuits can reveal confidential information or private cryptographic algorithms. However, the costs of these kinds of attacks is considerably high. Moreover, during manufacturing of the cards, measures are taken to prevent tampering with the cards.
  - *Logical attacks.* Using inappropriate supply voltages or temperatures, some chips can leak confidential information.
  - *Side channel attacks.* An adversary can exploit information leaked by the physical characteristics of the card during execution of the algorithm. Extra information can be deduced by analysing the timing of operations, power consumption, or radiation. An example is power consumption analysis and electromagnetic radiation analysis to extract cryptographic keys.
  - *Typical attacks with contactless cards.* Eavesdropping, interruption of operations (interference), denial of service, ...

Privacy

- Pro:
  - *PIN protection.* Confidential information can be protected with a PIN code.
- Contra:
  - *Data collection.* Bad implementations of smart cards release more personal information than required by the application.

Usability/Userfriendliness

- Pro:
  - *Acceptance rate.* Smart cards are widely accepted.
  - *Convenience.* The use of smart cards is user-friendly and fast (if no PIN is required).
- Contra:
  - PIN codes must be remembered.

Availability/Mobility

Contact cards

- Pro:
  - *Comfort.* Easy to carry in wallets, ...
  - *Adoption.* There is already a widespread acceptance, usage and application of smart card technology.

  - *Infrastructure.* Transactions can be done offline. No network connection is required.

- Contra:

  - *Loss.* Cards that are left in the card reader are not uncommon.
  - *Infrastructure.* Requires a card acceptance device (CAD).
  - *Failure rate.* The plastic housing in which the chip is embedded is fairly flexible. When bending the card, the chip can break.
  - *Damage.* Static electricity on the contacts may damage the chip. Moreover, contamination and contact wear can cause reading/writing errors.
  - *Electromechanical problems.* In mobile equipment, vibrations can cause failures.

Contactless cards

- Pro:

  - *Damage.* No problem with contacts that wear out, or with static electricity.
  - *Comfort.* Easy to carry in wallets, ...
  - *Adoption.* There is already a widespread acceptance, usage and application of contactless smart card technology.

Instances of technology.

- Credit cards / Bank cards / Proton cards (electronic purses)
- SIM cards
- Loyalty cards
- Healthcare cards
- Fast ticketing in public transport (subway, train), parking and road tolling
- Identification cards (e.g. at the library, canteen, vending machines and other services)

Development environments.

- Java smart card Framework
- Windows for smart cards (.NET cards)
- Vendor specific development tools

3.1.2 Communication technologies

Infrared communication protocol

Description of technology.
The Infrared Data Association (IrDA) defines communication protocol standards for infrared wireless communication. Infrared communication requires direct line-of-sight. Nowadays it is replaced by Bluetooth communication but it is still used in environments where interference makes RF communication infeasible.

Evaluation of technology.

Security

- Pro:
  - *Line-of-sight.* Because of the line-of-sight requirement, the customer knows with whom he is communicating with.
  - *Range.* Low distance (< 2m) and limited field of sight.
  - *Eavesdropping.* High degree of control over data leakage since infrared cannot permeate walls.
- Contra:
  - *Cryptography.* The IrDA standard does not specify any security measures for data transfer. It does not provide any link-level security, so there is no authentication or authorisation, and all information is sent unencrypted.
  - *Eavesdropping.* Eavesdropping on a communication channel is possible by detecting reflected infrared-light.

Privacy

- Pro:
  - *Line-of-sight.* Line-of-sight is required for data transfer.

Usability/Userfriendliness

- Pro:
  - *Convenience.* Half-duplex connection up to 115.2kbps (low cost), high-speed extensions up to 16Mbps (more expensive)
- Contra:
  - *Adoption.* No widespread use.

Availability/Mobility

- Pro:
  - *Power consumption.* Infrared components have a low power consumption.
  - *Adoption.* Infrared is available in many devices such as laptops, PDAs, GSMs. However, in newer systems it is often replaced by Bluetooth (see further).
  - *Costs.* Low cost of communication components.
  - *Interference.* Tolerance to interference caused by third-parties.
- Contra:
  - *Line-of-sight.* Infrared cannot permeate walls or physical barriers.
  - *Range.* Low distance

Instances of technology.

- Data transfers
- Remote control
- Printing

Bluetooth

Description of technology.
Bluetooth is an open industry standard for short-range wireless communication of voice and data, permitting ad hoc networking.

Evaluation of technology.

Security

- Pro:
  - *Cryptography.* The specification contains an authentication and encryption system. A secure Bluetooth connection requires authorisation and authentication before data is accepted.
  - *Tracking.* Bluetooth can be switched off to disable device tracking.
  - *Scanning.* Bluetooth devices can be set in invisible mode.
  - *Selectivity.* Bluetooth devices only respond to known devices.
- Contra:
  - *Awareness.* Many users are unaware of the security risks involved. Only the name of an access point is shown when pairing. A fake Bluetooth access point can pretend to be a trusted device.

  - *Cryptography.* Encryption is only optional in Bluetooth and has a number of vulnerabilities. Weak PINs can be guessed. Unit keys are insecure (keys are generated the first time a device is used and do not change thereafter). The quality of the random number generator can be low resulting in weak encryption.
  - *Defaults.* Insecure default settings.
  - *Integrity.* Weak protection of integrity.
  - *Quality of implementation.* Many implementation bugs are found in various devices. For instance authentication and authorisation is not always correctly implemented by manufacturers.
  - *Bluejacking.* Once a device is accepted, the connection is automatically set-up since the contact is 'known/trusted'.
  - *Bluebugging.* Based on implementation bugs, adversaries can harm the system.
  - *Bluespam.* Spam messages can be sent to Bluetooth enabled devices
  - *Car Whisperer.* Some attacks allow adversaries to send and receive audio from a Bluetooth-enabled car stereo.

Privacy

- Pro:
  - Bluetooth connections can be switched of, disabling tracking.
- Contra:
  - *Hiding.* If Bluetooth devices are set to non-discoverable/invisible mode, they can still be discovered.
  - *Identity.* The address of a Bluetooth device, which can be used as an identifier, is not encrypted.
  - *Personal.* Bluetooth devices such as mobiles and PDAs often contain sensitive personal data.
  - *Tracking.* Individual devices can be tracked

Usability/Userfriendliness

- Pro:
  - *Convenience.* Easy setup of a Bluetooth connection (automatic) and simple data transfer.
- Contra:
  - *Awareness.* The possibilities of Bluetooth are not widely known.

Availability/Mobility.

- Pro:
  - *Wireless.* Bluetooth is a wireless technology.
  - *Costs.* Bluetooth is inexpensive and available in many devices.
  - *Distance.* High-gain antennas could stretch communications to 90 meters.
- Contra:
  - *Interference.* Wireless RF communication can be disturbed by other RF systems.

Instances of technology.

- Wireless connection of office peripherals.
- Three way phones: At home, your phone functions as a portable phone (fixed line charge). Outside, it functions as a mobile phone (cellular charge). And when your phone comes within range of another mobile phone with built-in Bluetooth technology it functions as a walkie-talkie (no charge).
- Transfer of documents with selected participants.
- Connecting to wireless headsets.
- Automatic synchronisation between desktop, PDA, phone, ...

WLAN

Description of technology.
A wireless local area network or WLAN is a wireless version of the local area network, allowing customers to link computers with each other without wires.

Evaluation of technology.

Security

- Pro:
  - *Cryptography.* Encrypted communication is possible.
  - *Availability.* A wireless device can be switched off.
- Contra:
  - *Cryptography.* WEP is insecure. WPA/WPA-2 is more secure, but vulnerable to DoS attacks. Management frames containing valuable information are sent in clear text.
  - *WarDriving.* Armed with a laptop and a good Wi-Fi antenna, adversaries drive around to find hotspots and compromise the underlaying system, allowing them to have unauthorized access.

Privacy

- Pro:
  - The strong encryption schemes allow for confidential communication. In home networks, only local connections can be traced.
- Contra:
  - Open access points allow information/identity theft.
  - Public hotspots allow profiling and tracking.
  - Public hotspots allow directed attacks.

Usability/Userfriendliness

- Pro:
  - *Convenience.* Ease-of-use principle, easy installation.
- Contra:
  - *Awareness.* Ordinary users don't know the security risks involved.

Availability/Mobility

- Pro:
  - *Wireless.* Wireless communication, often available 24/24.
  - *Adoption.* Wi-Fi becomes a common, well known technology widely adopted for all kinds of networks.
- Contra:
  - *Denial of Service.* Attacks such as denial of service during the 4-Way Handshake, could prevent access to the network.

Instances of technology.

- Wireless home networks.
- Public businesses such as coffee shops or malls offer wireless access to their customers.
- Large wireless network projects put up in many major cities.
- Remote printing.

NFC technology

Description of technology.
Near Field Communication (NFC (ISO 18092)) is a short-range wireless technology standard, based on RFID, designed for intuitive, simple and safe communication between electronic devices.

Evaluation of technology.

Security

- Pro:
  - *Resources.* The rich processing capabilities and resources of an NFC device (such as a mobile phone) can be used to implement complex and strong security mechanisms.
  - *Secure channel.* A secure channel can solve most of the communication problems.
  - *Man-in-the-middle-attack.* The short-range decreases this vulnerability
- Contra:
  - *Secure storage.* Since the NFC enabled device is possibly connected with other systems, additional security measures may be required to secure the device.
  - *Eavesdropping.* Eavesdropping up to $10m$ and more is possible.
  - *Denial of service.* Interference during communication can be used for DoS attacks.
  - *Integrity.* Data can be modified/inserted during transmission.

Privacy

- *Secure development.* Guidelines for NFC ecosystem participants will give vendors and providers tools to ensure privacy for consumers.

Usability/Userfriendliness

- *Convenience.* Easy to use. Just hold your device close enough to another device.

Availability/Mobility

- Pro:
  - *Setup.* NFC enables easy set-up and intuitive interactions (pairing for Bluetooth and Wi-Fi).
  - *Contactless transactions.* NFC supports contactless transactions (compatible with ISO 14443 A, B and C)
  - *Power consumption.* NFC is both active and passive (interacts with RFID) depending on the application.

- Contra:
  - *Adoption.* A broad market deployment of NFC technology will take some years. A recent study by ABI Research projects that 450 million mobile phones will be NFC-enabled by 2011.

Instances of technology.
Applications of NFC technology include contactless transactions such as payment and transit ticketing, simple and fast data transfers such as calendar synchronisation or electronic business cards and access to online digital content. A wide range of devices and machines are likely to become NFC enabled. Here are some examples:

- Mobile phones
- Vending machines
- Parking meters
- Check-out cash registers or point-of-sale equipment
- ATMs
- Office, house and garage doors
- Personal computers
- Posters, street signs, bus stops, local points of interest (with NFC-readable tags only)
- Product packaging

3.1.3 Security support on mobile devices

.NET Compact Framework enabled devices

Description of technology.
.NET Compact Framework (CF) is a reduced/specialised version of the .NET Framework to run on Windows CE based devices such as smartphones, PDAs, etc. It contains some of the classes available in the full .NET Framework, extended with some libraries specific for mobile devices. The information below is valid for [Compact Framework 2.0 and above] and in some cases only for Windows Mobile (WM)

Evaluation of technology.

Communication

- *Serial communication (IrDA, Bluetooth).* Supported by the framework.
- *TCP/IP & UDP.* Supported by the framework.
- *SMS.* Supported by the framework (WM).
- *WebServices.* Supported by the framework.

Security

- *Cryptography.* The following features are supported: X.509 certificates, hashing (MD5 and SHA1), symmetric key encryption (RC2, RC4, 3DES, DES), asymmetric key encryption (RSA, DSA).
- *Secure communication.* The .NET CF includes support for SSL.
- *SIM.* Available through native calls (WM).

References.

- How to Intercept Incoming Short Message System (SMS) Messages
  url: `http://msdn2.microsoft.com/en-gb/library/bb932385.aspx`
- Securing Communications with SSL and the .NET Compact Framework
  url: `http://msdn2.microsoft.com/en-us/library/bb738067.aspx`
- SIM Manager Extended Phone Book Support
  url: `http://msdn2.microsoft.com/en-gb/library/bb787183.aspx`
- SIM Programming with the .NET Compact Framework
  url: `http://msdn2.microsoft.com/en-gb/library/ms839358.aspx`
- Application and Network Authentication with the .NET Compact Framework
  url: `http://msdn2.microsoft.com/en-gb/library/bb736226.aspx`
- Refer to the full .NET Framework on how to make use of the Compact Framework.

Java enabled devices

Description of technology.
Java Platform, Micro Edition (Java ME) provides a robust, flexible environment for portable applications running on mobile and other embedded devices such as mobile phones, PDAs, etc. Java ME contains flexible user interfaces, robust security, built-in network protocols and support for networked and offline applications. The evaluation below focuses on high-end consumer devices (such as smartphones and PDAs). Specifications of low-end consumer devices (such as GSM) have to be studied case-by-case. More information can be found in the references.
Remark: The capabilities of Java ME implementations strongly depend on the JSRs (specifications) implemented on the device.

Evaluation of technology.

Communication

- *Serial communication (IrDA, Bluetooth).* Supported by Java ME.
- *TCP/IP & UDP.* Supported by Java ME.
- *SMS.* Supported by Java ME (also for Low-end devices).
- *WebServices.* Supported by Java ME (also for Low-end devices).

Security

- *Cryptography.* The following features are supported: X.509 certificates, Hashing, Symmetric and asymmetric key encryption. Implements most of the functionalities that are available in Java SE.
- *Secure communication.* Support for SSL.
- *SIM.* Supported by Java ME (also for Low-end devices).

References.

- Information on the capabilities of Java enabled devices
  url: `http://ingenieur.kahosl.be/projecten/advies/pw/`
- Documentation on the Java Platform, Micro Edition (Java ME)
  url: `http://java.sun.com/javame/reference/apis.jsp`
- Low-end Consumer Devices: Java Technology for the Wireless Industry (JTWI) (JSR 185) gives a clear specification for device manufacturers, operators, and application developers on what is available on the Java enabled devices.
  url: `http://developers.sun.com/mobility/midp/articles/jtwi/`
- The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms.
  url: `http://www.bouncycastle.org/specifications.html`
- Security and Trust Services API for J2ME (SATSA)
  url: `http://java.sun.com/products/satsa/`
- How to Encrypt Data in Java ME Applications Using the SATSA API (only devices supporting the Mobile Service Architecture)
  url: `http://blogs.sun.com/mobility_techtips/entry/how_to_encrypt_data_in`

3.2 Cryptographic building blocks

3.2.1 Basic cryptographic primitives

In this section, important cryptographic building blocks are introduced that will be used in the three applications: e-ticketing, e-domotics and e-holiday park.

One-way functions and hash functions

A function f from a set X to a set Y is called a *one-way function* if $f(x)$ is "easy" to compute for all $x \in X$ but for most elements $y \in Image(f)$ it is "computationally infeasible" to find any $x \in X$ such that $f(x) = y$.
An example of such a one-way function is $a^x \bmod p$ where p is a large prime number and $a \in \mathbb{Z}_p$.

A *trapdoor one-way function* f is a one-way function with the additional property that given some extra information (called the *trapdoor information*, it becomes feasible to find for any given $y \in Image(f)$, an $x \in X$ such that $f(x) = y$.
An example of such a function is $x^3 \bmod n$ where n is the product of two large prime numbers (p and q). If the factors of n are unknown and large, it is difficult to find an x for which $x^3 \bmod n = y$. However, if the factors p and q of n are known, there exists an efficient algorithm for computing modular cubes.

A *hash function* h is a computationally efficient function mapping binary strings of arbitrary length to binary strings of a fixed length, called *hash-values*. For n-bit hash-values, the probability that a randomly chosen string gets mapped to a particular n-bit hash-value is 2^{-n}; hence, a hash-value serves as a compact representative of an input string. Usually, a hash function h is chosen such that:

- it is computationally infeasible to find two distinct inputs x_1 and x_2 which hash to the same value: $h(x_1) = h(x_2)$. [**collision free**]
- given a specific hash-value y, it is computationally infeasible to find an input x such that $h(x) = y$. [**one-wayness**]

Common hash functions are SHA1 (160-bit hash-values), MD5 (128-bit hash-values) and RIPE-MD160 (160-bit hash-values). Note that recently, some collisions for MD5 have been constructed.

Usage of hash functions and one-way functions

Hash functions are typically used in signature schemes, where the signer signs the hash-value (i.e. the signer will sign the hash-value of the data to be signed instead of the data itself). This way, the computation of the signature can be made more efficient and the size of the signature is limited in length. Note that it is important that the hash function is collision free and one-way.

One-way functions or hash functions can also be used *to hide personal information inside certified data* (i.e. data signed by a certification authority (CA); e.g. a certificate [cfr. section 3.2.1]). The owner of the certified data can show it to another party and disclose one or more values that are hidden in the signed data. Hence, the owner can decide which personal information to disclose and which information to keep hidden.

Figure 3.1 shows a signed statement[1] that contains three plain text values: pseudonym, policy and class and three hash-values, namely of the name, of the birth date and of the address of the owner. R_{name}, R_{date} and $R_{address}$ are random values that prevent dictionary attacks[2]. Note that if the holder wishes to disclose his certified address, he needs to disclose both his address and the random value $R_{address}$. His name and birth date will remain hidden for the receiver of the signed statement.

Pseudonym	0x123456
Policy	URI="//www.ca.com/certification-policy.htm"
Class	VIP status
Name	SHA1("John Doe" || R_{name})
Birth date	SHA1("1968/10/07" || R_{date})
Address	SHA1("Church street 1, 777777 Utopia" || $R_{address}$)
Signature	0x0A1B2C3D...

Figure 3.1: Certified hashed data

Note that in figure 3.1, another hash-value was used to generate the signature:

$$\text{Signature} \leftarrow \text{sign}(\text{PrivateKey}_{CA}, \text{RIPE-MD160}\{\text{Pseudonym} \parallel \ldots \parallel \text{Address}\})$$

Symmetric-Key Cryptography

Symmetric-Key cryptography, also called *conventional* or *secret-key cryptography*, is based on a secret key that is shared among the parties that want to exchange confidential data. The *same* key is used both for encryption and decryption (or it is easy to derive the decryption key from the encryption key).

- *Ciphertext ← encrypt (Key, Plaintext)*
  The *encrypt*-operation converts *Plaintext* into *Ciphertext* using the secret key *Key*.
- *Plaintext ← decrypt (Key, Ciphertext)*
  The *decrypt*-operation converts *Ciphertext* into *Plaintext* using the secret key *Key*.

Every encryption scheme can be turned into a *probabilistic encryption scheme* by adding random data to the plain text. This way, different encryptions of the same plain text (with the same key) will yield a different cipher text[3]. When the cipher text is decrypted, the random value is discarded.

- *Ciphertext ← encrypt (Key, { Random || Plaintext})*
  Before encrypting the plain text, a random value is generated and concatenated with the plain text.

[1] The statement is signed by a certification authority. The policy should explain how the CA has verified the certified values.

[2] Although the hash function h is one-way, it is still possible to try different $x \in X$ values to see whether $h(x) = y$. If the set X has only a limited number of elements, a dictionary attack is easy to perform.

[3] Hence, a passive attacker cannot link both cipher texts.

- *(Random || Plaintext) ← decrypt (Key, Ciphertext)*
  The decryption of the cipher text yields both a random value and the plain text; the random value is discarded.

Secret key cryptography can be used to *conditionally hide personal information* (e.g. the identity of a person or another property) : i.e. the holder of the key can decrypt the cipher text containing the hidden information and reveal it in case of abuse or liability issues. For example, a pseudonym can be derived from the identity of a person through a probabilistic encryption with a key, K_{TTP}, known by a trusted third party (TTP):

$$PN = encrypt(K_{TTP}, \{RND \mid\mid ID\}) \tag{3.1}$$

where PN represents the pseudonym, RND a random value and ID the identity information. In this case, the secret key K_{TTP} is only known to the TTP. In case of abuse or liability, the TTP can decrypt the pseudonym and reveal the true identity of the party involved.

Public-Key Cryptography

In public-key encryption schemes, each entity A has a *public key* e and a corresponding *private key* d. In secure systems, computing d given e is computationally infeasible. Two operations are defined:

- *Ciphertext ← encrypt (PublicKey, Plaintext)*
  The *encrypt*-operation converts *Plaintext* into *Ciphertext* using the public key *PublicKey*.

- *Plaintext ← decrypt (PrivateKey, Ciphertext)*
  The *decrypt*-operation converts *Ciphertext* into *Plaintext* using the private key *PrivateKey*.

Examples of public-key encryption schemes are RSA, Rabin, ElGamal, McEliece.

Any entity B wishing to send a message m to A, first obtains an *authentic copy* of A's public key e, uses the encryption algorithm to compute the cipher text $c = encrypt(e, m)$ and transmits c to A. To decrypt c, A applies the decryption algorithm to obtain the original message $m = decrypt(d, c)$.

The public key need not to be kept secret and, in fact, should be made public. Only its authenticity is required to guarantee that A is the only party who knows the corresponding private key. A primary advantage of such systems is that providing authentic public keys is in general easier to realize than distributing secret keys securely (as required in symmetric-key systems). In a large-scale networked environment it is impossible to guarantee that prior relationships between communicating entities have been established or that a trusted repository exists with all used public keys. Therefore, certificates [48] will establish the authentic binding between public key and owner of the corresponding private key (cfr. section 3.2.1).

Some public-key encryption schemes also allow for digitally signing of data. In this case, two more operations are defined:

- *Sig ← sign(PrivateKey, Data)*
  The *sign*-operation generates a signature *Sig* on message *Data* using the private key *PrivateKey*.

- *SigOK* ← *verify(PublicKey, Sig, Data)*
  The *verify*-operation verifies whether the signature *Sig* on *Data* is genuine. The verification uses the public key *PublicKey* that corresponds to the private key with which the signature was generated.

Public-Key Certificates

A **public-key certificate** (also called "certificate") is a digitally signed statement from one entity, *the certification authority* (CA), saying that the **public key** (and some other information) of another entity has some specific value. A CA acts as a Trusted Third Party. CAs are entities (e.g., businesses) that are trusted to sign (issue) certificates for other entities. It is assumed that CAs will only create valid and reliable certificates as they are bound by legal agreements. There are many public Certification Authorities, such as VeriSign, Thawte, Entrust, and so on. An organization can also run its own Certification Authority.

A certificate binds a public key held by an entity (such as person, organization, account, device, or site) to a set of information that identifies the entity that is the holder of the corresponding private key. In most cases involving *identity certificates*, this entity is known as the *subject* or *subscriber* of the certificate. An exception, however, are *anonymous certificates* (in which the identity of the individual or organization is not available from the certificate itself). Other types of certificates bind public keys to attributes of an entity other than the entity's identity, such as a role, a title, or creditworthiness.

A certificate is used by a *certificate user* or *relying party* that needs to use, and rely upon the accuracy of, the binding between the subject's public key distributed via that certificate and the identity and/or other attributes of the subject contained in that certificate. A relying party is an entity that uses a public key in a certificate (for signature verification or encryption). Frequently, it is an entity that verifies a digital signature from the certificate's subject where the digital signature is associated with an email, web form, electronic document, or other data. Other examples of relying parties can include a sender of encrypted email to the subscriber, a user of a web browser relying on a server certificate during a secure sockets layer (SSL) session, and an entity operating a server that controls access to online information using client certificates as an access control mechanism.

The degree to which a relying party can trust the binding embodied in a certificate depends on several factors. These factors can include:

- the practices followed by the certification authority (CA) in authenticating the subject;
- the CA's operating policy, procedures, and security controls;
- the scope of the subscriber's responsibilities (for example, in protecting the private key);
- the stated responsibilities and liability terms and conditions of the CA (for example, warranties, disclaimers of warranties, and limitations of liability).

The certificate may contain a field declaring that one or more specific certificate policies [52] apply to that certificate. The policy may be used by a relying party to help in deciding whether a certificate, and the binding therein, are sufficiently trustworthy and otherwise appropriate for a particular application.

X.509 Certificates X.509 certificates [48, 47] were defined in 1988 and modified in 1993 (yielding Version 2 certificates); an extension field was added in 1995 (yielding Version 3 certificates). Standard extensions for Version 3 certificates appear in an amendment to X.509. These certificates accommodate information related to key identifiers, key usage, certificate policy, alternate names (versus X.500 names) and name attributes, certification path constraints, and enhancements for certificate revocation including revocation reasons and CRL partitioning. A typical X.509 certificate has the following structure:

Version
Serial Number
Algorithm ID
Issuer
Validity – Not Before – Not After
Subject
Subject Public Key Info
Public Key Algorithm
Subject Public Key
Issuer Unique Identifier (Optional)
Subject Unique Identifier (Optional)
Extensions (Optional) ...
Certificate Signature Algorithm
Certificate Signature

An example X.509 certificate is shown in figure 3.2.

Certificate revocation

A **certificate revocation list** (CRL) [47] is a list of certificates (more accurately: their serial numbers) which have been revoked, are no longer valid, and should not be relied on by any system entity.

There are different revocation reasons defined in RFC 3280:

revoked: A certificate is irreversibly revoked (and entered on a CRL) if, for instance, it is discovered that the certificate authority (CA) had improperly issued a certificate or a privatekey is thought to have been compromised. Certificates may also be revoked for failure of the identified entity to adhere to policy requirements such as publication of false documents, misrepresentation of software behaviour, or violation of any other policy specified by the CA operator or its customer. The most common reason for revocation is the user's not being in sole possession of the private key (e.g token containing the private key has been lost or stolen).

hold/supended: This reversible status can be used to notice the temporary invalidity of the certificate, for instance when the user is not sure if the private key has been lost. If, in

this example, the private key was found again and nobody had access to it, the status can be reinstated, and the certificate is valid again, thus removing the certificate from further CRLs.
Certificates in the Belgian e-ID card are initially set to the **suspended**-status, until the card has been issued to and activated by the card holder.

Usually, a CRL is generated after a clearly defined time frame and (optionally) immediately after a certificate has been revoked. The CRL is always issued by the same CA that issued the corresponding certificates. All CRLs have a (often short) lifetime in which they are valid and in which they may be consulted by a PKI-enabled application to verify a counterpart's certificate prior its use. To prevent spoofing or denial-of-service attacks, CRLs are usually signed by the issuing CA.

Best practices require that a certificate's validity must be checked whenever one wants to rely on that certificate. Failing this, a revoked certificate may be incorrectly accepted as valid. This means that to use a PKI effectively one must have access to current CRLs (i.e. Internet access in the case of a PKI).

An alternative to using CRLs, which is especially useful for software clients, is the on-line certificate validation protocol **Online Certificate Status Protocol** (OCSP) [43]. OCSP has the primary benefit of requiring less network bandwidth and thus enabling real-time and near real-time status checks for high volume or high value operations. It is an Internet protocol used for obtaining the revocation status of an X.509 digital certificate. It is described in RFC 2560. Compared to CRLs, OCSP has the following advantages and disadvantages:

- Since an OCSP response contains less information than a typical CRL, OCSP can feasibly provide more timely information regarding the revocation status of a certificate without burdening the network. However, the greater number of requests and connection overhead may overwhelm this benefit if the client does not cache responses.

- Using OCSP, clients do not need to parse CRLs themselves, saving client-side complexity. However, this is balanced by the practical need to maintain a cache. In practice, such considerations are of little consequence, since most applications rely on third-party libraries for all X.509 functions.

- CRLs may be seen as analogous to a credit card company's "bad customer list" – an unnecessary public exposure.

- OCSP discloses to the responder that a particular network host used a particular certificate at a particular time. OCSP does not mandate encryption, so this information may also be intercepted by other parties.

Pseudonym Certificates

In pseudonym certificates, the identity of the certificate holder is replaced by a pseudonym.

The pseudonym can be a random number (in fact, the public key itself can be used as a pseudonym), or it can be derived from the true identity of the certificate holder. Two schemes are often used:

- $PN = hash(ID \mid\mid RD)$
  The pseudonym PN is a hash-value of the true identity ID concatenated with a ran-

dom value RD. In case of disputes, the pseudonym owner can easily prove that the pseudonym belongs to him by revealing both the identity and the random number.

- $PN = encrypt(Key_{TTP}, \{ID \mid\mid RD\})$
  The pseudonym PN is the encryption of the ID with the public key of a TTP. If the pseudonym is generated by the TTP, a conventional encryption scheme can be used. In case of dispute or abuse, the TTP can easily reveal the identity that corresponds with the pseudonym.

Note that all transactions in which the same certificate is used, can be linked to each other, and hence, user profiles can be assembled.

Additional attributes (besides identity and public key) can be embedded in a certificate. For instance, address information, birth date, etc. could be certified in a certificate. These additional attributes allow services to implement better access controls and better customization. However, if these attributes are not required in the current transaction, unnecessary personal information is revealed. Therefore, the CA can conceal the actual values of these attributes, by substituting the attributes values by their hash values. The certificate owner can then decide which additional attributes to reveal in each transaction. See also figure 3.1. In this case the certified data block represents a certificate.

3.2.2 Advanced building blocks

In this section a few advanced building blocks are briefly described: commitments, proofs of knowledge and credentials. They allow for fine-tuned disclosure of personal data.

Commitments

A **commitment** can be seen as the digital analogue of a "non-transparent sealed envelope". It enables a committer to hide a set of attributes (non-transparency property), while at the same time preventing him from changing these values after commitment (sealed property). The committing party can prove properties of the attributes embedded in the commitment.

- *(Com, OpenInfo) ← commit(Attribute(s))*
  A new commitment *Com* is generated as well as secret information *OpenInfo* containing, among others, the attributes embedded in *Com*. This information can be used to prove properties about the attributes.

- $\mathcal{P}\leftarrow \mathcal{V}$*: comProve(OpenInfo, Com, pred(attrs))*
  The public input to this protocol is both a commitment *Com* and a boolean predicate *pred* concerning *com*'s attributes. If $\mathcal{V}$ accepts the proof, $\mathcal{V}$ is convinced that $\mathcal{P}$ knows *OpenInfo* belonging to *Com*, and that *Com*'s attributes satisfy predicate *pred*. Note that *comProve* is an interactive protocol.

A simple example of an application of commitments is secure coin-flipping. Suppose, Bob and Alice want to resolve some dilemma by coin flipping. If they are physically in the same place, the procedure would be (1) Bob "calls" the coin flip, (2) Alice flips the coin and (3) if Bob's call is correct, he wins, otherwise Alice wins. If they are not in the same place, however, this does not work, since Bob has to trust Alice to report the outcome of the coin flip correctly. With commitments, a similar procedure can be constructed: (1) Bob "calls"

the coin flip but only tells Alice a commitment to the call, (2) Alice flips the coin and reports the result to Bob, (3) Bob reveals what he committed to, (4) if Bob's call matches the result, Bob wins, otherwise, Alice wins.

Proof of knowledge and Zero-knowledge proof

A *proof of knowledge* is an interactive proof in which the prover succeeds 'convincing' a verifier that he knows something. Assume a set X of publicly known elements; for each $x \in X$, there exists a witness $w \in W$ for which the relation $R(x, w)$ is true. The prover $\mathcal{P}$ can then prove to $\mathcal{V}$ that he knows w without revealing it to $\mathcal{V}$.

$$\mathcal{P} \leftarrow \mathcal{V} : PK\{(w) : R(x, w)\} \tag{3.2}$$

where x is the public input to the protocol, and w the corresponding secret witness w, the knowledge of which is proven by $\mathcal{P}$ to $\mathcal{V}$.

A *zero-knowledge proof* is an interactive protocol in which a party $\mathcal{P}$ proves to another party $\mathcal{V}$ that a statement is true, without revealing anything other than the veracity of the statement.

$$\mathcal{P} \leftarrow \mathcal{V} : ZKP\{(x) : pred(x)\} \tag{3.3}$$

where x is a secret value of which predicate *pred* is proven to $\mathcal{V}$. $\mathcal{V}$ does not gain any other information about x except that $pred(x)$ is true.

Example of Proof of knowledge and Zero-Knowledge proof An authentication protocol based on public key cryptography usually uses a proof-of-knowledge protocol, in which the prover proves that he knows the private key:

$\mathcal{P}$		$\mathcal{V}$
		choose random challenge *rnd*
		$c \leftarrow$ encrypt(PubKey, *rnd*)
	$\xleftarrow{c}$	
$r \leftarrow$ decrypt(PrivKey, c)		
	$\xrightarrow{r}$	
		if ($r == rnd$)
		then SUCCESSFUL
		else FAILED

$\mathcal{P}$ proves that he knows the private key (PrivKey) that corresponds to the public key (PubKey). This is abbreviated as follows:

$$\mathcal{P} \rightarrow \mathcal{V}\text{: PK}\{\text{(PrivKey): } rnd = \text{decrypt(PrivKey, encrypt(PubKey, } rnd\text{))}\}$$

Although the private key (PrivKey) is not revealed to $\mathcal{V}$, a dishonest $\mathcal{V}$ can learn something about the private key by sending random data to $\mathcal{P}$ and have $\mathcal{P}$ decrypt that data ($\mathcal{P}$ is used as an oracle by $\mathcal{V}$ for decrypting data).

The previous protocol can easily be transformed into a zero-knowledge proof:

$\mathcal{P}$		$\mathcal{V}$
		choose random challenge *rnd* $c \leftarrow$ encrypt(PubKey, *rnd*)
	$\xleftarrow{c}$	
$r \leftarrow$ decrypt(PrivKey, c) *(com, openinfo)* $\leftarrow$ commit(r)		
	$\xrightarrow{com}$	
	$\xleftarrow{rnd}$	
if ($rnd \neq r$) then ABORT		
	$\xrightarrow{openinfo}$	
		$r \leftarrow$ openCommit(*com, openinfo*) if ($r == rnd$) then SUCCESSFUL else FAILED

Here $\mathcal{P}$ first sends a commitment to the challenge, and when $\mathcal{V}$ reveals the challenge, $\mathcal{P}$ opens the commitment for $\mathcal{V}$. A dishonest $\mathcal{V}$ is not able to send the challenge in cleartext (in the third message) if the first message was random data.

$\mathcal{P}$ proves (in zero-knowledge) that he knows the private key (PrivKey) that corresponds to the public key (PubKey). This is abbreviated as follows:

$$\mathcal{P} \rightarrow \mathcal{V}\text{: ZKP}\{\text{(PrivKey): } rnd = \text{decrypt(PrivKey, encrypt(PubKey, } rnd\text{))}\}$$

Other examples of zero-knowledge proves include proving that an attribute embedded in a commitment or credential (cfr. section 3.2.3), is greater than a certain value (e.g. the age-attribute is greater than 18) or that the attribute lies in a fixed interval (e.g. the age-attribute lies in [18,35]).

Verifiable encryptions

Verifiable encryptions have all the characteristics of regular (public key) encryptions. Additionally, they enable the creator $\mathcal{P}$ to demonstrate properties of the encrypted plain text. As an example, $\mathcal{P}$ can prove to $\mathcal{V}$ that the encrypted plain text is encoded as an attribute in a commitment or credential.

$$c = VE(PubKey, x) \tag{3.4}$$

c refers to the cipher text, x refers to the plain text, $PubKey$ is the public key of a TTP. Note that the corresponding private key may not be known to $\mathcal{V}$ nor $\mathcal{P}$.

Verifiable encryptions can be used to implement accountability in anonymous transactions. The customer will send a verifiable encryption of his pseudonym (or identity) to the service provider, thereby proving that the encrypted plain text is correctly formed (e.g. is equal to the pseudonym embedded in a credential). The service provider is assured that when abuse is detected, the TTP will be able to reveal the pseudonym (or identity) of the anonymous customer.

3.2.3 Anonymous credentials

Anonymous credential systems ([24], [23], [22]) allow for anonymous yet accountable transactions between users and organizations and allow for *selective disclosure* by showing properties of credential attributes (e.g. $age > 18$) while hiding all the other credential attribute information. In the Idemix system [23], different usages of the same credential are *unlinkable* (except when unique attribute values are revealed). Credentials can have features such as an expiry date, the allowed number of times it can be shown and the possibility to be revoked. A mix network ([46], [33]) is required to provide for anonymity at the network layer.

The most relevant anonymous credential protocols are:

- $U \leftrightarrows O$: (Nym, *Sig*) $\leftarrow$ generateSignedNym(Cert). One can establish multiple non-transferable pseudonyms (i.e. nyms) with the same organization. Here, the user signs the established Nym giving O a provable link between the nym and the identity certified in Cert.

- $U \leftarrow I$: *Cred* $\leftarrow$ issueCred(Nym, *attributes*). A credential is issued by I on a pseudonym Nym. The credential is known only to the user and cannot be shared. Also, a number of attributes, not necessarily known by I, is embedded into the credential.

- $U \leftrightarrows V$: transcript $\leftarrow$ authenticate(*Cred*, properties, [*DeanCond*], [*Msg*]). A user U authenticates to verifier V by proving possession of a valid credential *Cred*. U can selectively reveal credential attributes or properties thereof. The resulting transcript for V may be deanonymizable upon fulfillment of condition *DeanCond* (cfr. the deanonymize()). U may decide to sign a message *Msg* with his credential by a provable link between the transcript and the message. Different transcripts for the same credential are unlinkable (unless the value of a unique attribute is proved).credential's issue protocol.

- $U \rightarrow V$: prove(properties). Simplified notation of the above function. *Properties* will refer to the credential used in the proof.

- D: (Nym, *DeanProof*) $\leftarrow$ deanonymize(transcript, *condition*). If a credential show is deanonymizable, the pseudonym Nym on which the credential was issued can be revealed by a trusted deanonymizer D. $DeanProof$ proves the link between the transcript and the nym. D is only allowed to perform the deanonymization when *condition* fulfills *DeanCond* (included in the transcript).

Credential systems

There are currently two different credentials systems: Idemix[6] and U-Prove [18].

Idemix is still being implemented by IBM (currently, only a small part of the system is available) and will eventually become public domain. U-Prove is a commercial product of Credentica, which has recently been taken over by Microsoft. The major differences between the two systems are:

- In Idemix, different "shows" of the same credential are not linkable to each other, while in U-Prove these shows are linkable.

- Idemix uses accumulators for revocation (a revoked credential is removed from the accumulator), while U-Prove uses blacklists.

- Idemix has a deanonymization mechanism which allows for accountability in case of abuse.

The similarities between the systems can be summarized as follows:

- the issuer does not necessarily know all the values of the attributes embedded in a credential;
- a credential "show" cannot be linked to the credential "issue" protocol, even if the issuer and the verifier (of the credential show) collude, unless of course a unique attribute (e.g. a serial number) is disclosed during the "show".
- during a credential show protocol, the user can prove a fairly complex predicate (with AND- and OR-connectives).

3.3 Framework support

3.3.1 General overview

The main goal of the framework is to ease the use of different types of credentials and to improve the privacy of the user. The framework offers a generic interface to application developers. This interface allows to use complex technologies like Idemix. To realize such an architecture, the framework uses providers. A provider is a set of implementations that can be integrated into the framework. This approach allows to extend the framework with new technologies or to upgrade it with better implementations of already existing technologies.

An overview of the framework components can be found in figure 3.3. Each framework component has a handler and a manager interface. The handler interfaces offer methods to handle one specific item (credential/connection/dispute/etc.). The manager interfaces have methods to manage these items (e.g. choose the most appropriate credential/connection for a certain action w.r.t. security/privacy requirements). These interfaces are implemented by the providers. The applications are implemented on top of the framework.

3.3.2 Components of the framework

Communication Handler

The `CommunicationHandler` defines a generic interface to handle communication between two entities. This interface can be implemented by a provider. The framework provides the same modules and interfaces for both clients and servers. However, they use complementary methods. Different technologies that can be implemented are (SSL over) TCP sockets, Bluetooth, NFC, anonymous networks like Tor and JAP, etc. The `CommunicationManager` manages all connections in the framework. Depending on the requirements of the application (e.g. tamper proof, anonymous, integrity- and/or confidentiality-protected, etc.) the `CommunicationManager` can select the most appropriate `CommunicationHandler` implementation.

Credential Handler

The most important interface in the framework is the `CredentialHandler` interface. It defines a generic interface to perform some actions using credentials (like signing, authenticating, issuing credentials, etc.) while making abstraction of a specific technology (like X.509, Idemix, etc.). This interface can be implemented by several providers using different technologies. Each `CredentialHandler` implementation can only handle one type of credential (e.g. `CertificateCredentialHandler`, `PseudonymCertificateCredentialHandler`, `IdemixCredentialHandler`, `BelgianEidCredentialHandler`). The `CredentialManager` manages and selects the most appropriate credential(s) in the framework.

Interface This is a technical description of the interfaces that are currently defined in the `CredentialHandler`. This API is subject to change during the further development of the framework and as the development of the framework proceeds, the interface may evolve.

The following interfaces have been defined:

- authenticate: methods used to authenticate with a credential
- verifyAuthentication: methods used to verify the authentication
- sign: methods used to sign a message with a credential
- verifySignature: methods used to verify the signature of a message
- localSign: methods used to sign a message off-line with a credential
- getCredential: method used to request a credential
- issueCredential: methods used to issue a credential
- verifyTranscript: method used to check if an action is performed successfully.

authenticate The authenticate methods try to authenticate using a (set of) credential(s) over a connection to another party. During authentication, certain credential attributes may be revealed or proved (depending on the technology) to the server. This information is kept in the `ShowSpecification` argument. The server will request a proof of the required attributes, but the client can prove optional attributes. If the authentication is successful, an `AuthProverTranscript` object is returned. The latter contains information about the proof/release of credential attributes to the other party. If no `ShowSpecification` argument is provided, no credential attributes are revealed.

- `public AuthProverTranscript authenticate(Connection connection, Credential cred, ShowSpecification spec) throws CredentialActionException, ConnectionException`
- `public AuthProverTranscript authenticate(Connection connection, Credential[] creds, ShowSpecification spec) throws CredentialActionException, ConnectionException`
- `public AuthProverTranscript authenticate(Connection connection, Credential cred) throws CredentialActionException, ConnectionException`

- `public AuthProverTranscript authenticate(Connection connection, Credential[] creds) throws CredentialActionException, ConnectionException`

verifyAuthentication To respond to an authentication request the verifier needs to use the `verifyAuthenticate` method. The prover can prove/reveal the necessary credential attributes or properties thereof as specified by the verifier in the `ShowSpecification` argument. To define these restrictions, it is necessary to know which credentials will be used for each proof. This is specified in a `CredentialTemplate`. Each instance describes which arguments are kept in a specific type of credential. When the prover has authenticated successfully, an `AuthVerifierTranscript` is returned with proofs that were made during authentication and values of credential attributes that were revealed. If no `ShowSpecification` argument is used, no credential attributes are requested from the prover.

- `public AuthVerifierTranscript verifyAuthentication(Connection conn, CredentialTemplate credDesc, ShowSpecification spec) throws CredentialActionException, ConnectionException`
- `public AuthVerifierTranscript verifyAuthentication(Connection conn, CredentialTemplate[] credDescs, ShowSpecification spec) throws CredentialActionException, ConnectionException`

sign The `sign` methods sign a message using a (set of) credential(s). The signed message is sent over a connection to another party. Together with the creation of the signature, some credential attributes may be disclosed or properties thereof proved to the verifier. This information is kept in the `ShowSpecification` argument. If the signature is generated successfully, an `AuthProverTranscript` object is returned. The latter contains information about the message, the signature and the proof/release of credential attributes to the other party. If no `ShowSpecification` argument is passed, no credential attributes are revealed if possible.

verifySignature To receive a signed message from the signer with the `sign` method, the verifier must use the `verifySignature` method. This method checks whether the signature is correct, the requested (properties of) attributes are disclosed/proved as specified in the `ShowSpecification` argument. To construct the proper proofs it is necessary to know which credentials will be used. This is specified in the `CredentialTemplate` argument. These objects describe which arguments are available in a specific type of credential. When the signer successfully generated the signature, this method returns a `SignVerifierTranscript` with proofs or some revealed values of credential attributes together with the signature. If no `ShowSpecification` argument is provided, no credential attributes are requested from the signer if possible.

- `public SignVerifierTranscript verifySignature(Connection conn, CredentialTemplate credDesc, ShowSpecification spec) throws CredentialActionException, ConnectionException`
- `public SignVerifierTranscript verifySignature(Connection conn, CredentialTemplate[] credDescs, ShowSpecification spec) throws CredentialActionException, ConnectionException`

- `public SignProverTranscript sign(Connection connection, Credential cred, ShowSpecification spec, byte[] message) throws CredentialActionException, ConnectionException`
- `public SignProverTranscript sign(Connection connection, Credential[] creds, ShowSpecification spec, byte[] message) throws CredentialActionException, ConnectionException`
- `public SignProverTranscript sign(Connection connection, Credential cred, byte[] message) throws CredentialActionException, ConnectionException`
- `public SignProverTranscript sign(Connection connection, Credential[] creds, byte[] message) throws CredentialActionException, ConnectionException`

localSign This method is similar to the `sign` method, except that no connection is required when signing the message.

- `public SignProverTranscript localSign (Credential cred, ShowSpecification spec, byte[] message) throws CredentialActionException`
- `public SignProverTranscript localSign (Credential[] creds, ShowSpecification spec, byte[] message) throws CredentialActionException`
- `public SignProverTranscript localSign (Credential cred, byte[] message) throws CredentialActionException`
- `public SignProverTranscript localSign (Credential[] creds, byte[] message) throws CredentialActionException`

getCredential To access services from a service provider, it might be necessary to obtain a new credential issued by a credential issuer. The `CredentialHandler` provides an interface for retrieving credentials.

The `getCredential` method requests a new credential from another party over a `Connection`. The `CredentialTemplate` specifies the type and structure of the credential that the client wants to receive. In some cases, it is possible to define the value of some attributes in the new credential. These values can be sent to the issuing party by the `AttributeValues` object.

- `public ReceiveTranscript getCredential(Connection conn, CredentialTemplate template, AttributeValues values) throws CredentialActionException, ConnectionException`

issueCredential A credential issuer can use the `issueCredential` method to issue credentials to new users in response to the `getCredential` request. This method uses the same arguments as the `getCredential` method. There is only one extra argument `Credential` or `PrivateKey` from the issuer to sign the newly issued credential. The `AttributeValues` are the values that are certified by the issuer.

- `public IssueTranscript issueCredential(Connection conn, CredentialTemplate template, AttributeValues values, Credential issuerCred) throws CredentialActionException, ConnectionException`

- `public IssueTranscript issueCredential(Connection conn, CredentialTemplate template, AttributeValues values, PrivateKey issuerKey) throws CredentialActionException, ConnectionException`

verifyTranscript A `Transcript` object is returned when the `CredentialHandler` has performed an action successfully. This `Transcript` contains the evidence to check/proof afterwards that an action is performed successfully. This check is exactly what the `verifyTranscript` method does.

- `public boolean verifyTranscript(Transcript trans) throws CredentialActionException`

3.3.3 Providers

The `Handler` components in the framework only define a generic interface. An implementation is necessary before a component can be used. This implementation can be provided by `Provider` packages. After one or more providers are plugged into the framework, the managers can automatically select an implementation for each request. This subsection gives an overview of the classes in a `Provider` package that must be implemented to build a correct provider for the framework.

Provider class

Each provider package must implement the `Provider` interface which is defined in the framework. Currently, the interface consists of one method, namely `getImplementations`. The method returns a set of `Implementation` classes. An `Implementation` class keeps the class in the provider package that implements the `Handler` component of the framework. Moreover, it defines the technology that is used (e.g. Idemix, X.509, etc.). This information is used in the framework to forward API calls from applications to the corresponding classes (with the actual implementation) in the provider package.

A `Provider` that offers support for the Belgian eID, X.509 certificates can have the following structure:

```
public class Provider implements be.kuleuven.cs.idf.Provider {
  @Override
  public Implementation[] getImplementations() {
    return new Implementation[] {
      new Implementation(this, Component.CredentialHandler,
          X509CredentialHandler.class,
          new Technology("X509", new Version("0.0.1"))),
      new Implementation(this, Component.CredentialHandler,
          BelgianEidCredentialHandler.class,
          new Technology("BelgianEid", new Version("0.0.1"))),
      new Implementation(this, Component.CommunicationHandler,
          SocketCommunicationHandler.class,
          new Technology("Sockets", new Version("0.0.1")))
      };
  }
}
```

Listing 3.1: Example provider class

The classes `X509CredentialHandler`, `BelgianEidCredentialHandler` and `SocketCommuni-cationHandl` implement the corresponding `Handler` interface.

Implementing Handler class

The framework defines interfaces for different types of `Handlers`. To implement the generic interface for a specific `Handler` component, a class must be created that inherits the `Handler` component that will be implemented. An example of a `X509CredentialHandler` that implements the `CredentialHandler` interface is shown below.

```
import be.kuleuven.cs.idf.components.CredentialHandler;

public class X509CredentialHandler extends CredentialHandler {

  public X509CredentialHandler(Identity identity) {
    super(identity);
  }

  // implementation of CredentialHandler methods that are inherited

}
```

Listing 3.2: X.509 Credential Handler implementation

3.3.4 Example: simplified ticketing application

Developers can build secure applications using the identity framework. A simple ticketing application shows how the framework can be used.

Design

The ticketing application exists of 3 actors: the *buyer*, the *seller* of tickets (ticket shop) and the *issuer* of anonymous credentials (see section 3.2.3). The government issues Belgian eID cards. However, this phase is not within the scope of this example.

The sample application consists of 2 phases:

1. The buyer authenticates with his eID to the issuer. If authentication is successful and if the buyer didn't receive an anonymous credential yet, the issuer issues an anonymous credential to the buyer.

2. The buyer requests an event list to the ticket shop. To buy tickets, he sends a request to the ticket shop with the eventID and the number of tickets he wants to buy. If tickets are still available, the ticket shop sends a list of ticket numbers to the buyer. The buyer uses his anonymous credential to sign the ticket numbers. The ticket shop checks the signature and the validity of the anonymous credential.

Implementation

The application consists of three modules (i.e. one module per actor). First, common concepts are introduced (like a credential template, a transcript and an identity file). Next, the implementation of the *issuer* and the *ticket shop* are discussed. Finally, the *client module* is presented.

Credential Template A `CredentialTemplate` represents the structure of a credential. This structure defines the attributes that can/must be used and some technology-specific parameters for each type of credential. A `CredentialTemplate` can be seen as a credential without attribute values. An example XML file of a `CredentialTemplate` for a X.509 credential is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<CredentialTemplate Name="Ticket">
  <Security>
    <Technology Name="X509" Version="0.0.1"/>
    <Parameter Name="key_size" Type="java.lang.Integer">2048</Parameter>
    <Parameter Name="sign_algorithm" Type="java.lang.String">SHA1withRSA</
        Parameter>
  </Security>
  <Control>
    <Parameter Name="duration" Type="java.lang.Long">86400</Parameter>
    <Parameter Name="signing" Type="java.lang.Boolean">false</Parameter>
    <Parameter Name="authentication" Type="java.lang.Boolean">true</Parameter>
  </Control>
  <AttributeSpecs>
    <AttributeSpec Name="subject" Type="java.lang.String" Required="true"
        Source="Server"/>
  </AttributeSpecs>
</CredentialTemplate>
```

Listing 3.3: Example credential template

Transcript After a successful action, the `CredentialHandler` returns a `Transcript`. A *Transcript* contains evidence of the credential show and context information. Examples are challenges used in the protocol, the time the action is performed, the hash of the message that has been signed, properties or values of attributes that have been disclosed, ... The `Transcript` class is a superclass. Each action in the `CredentialHandler` returns a class that inherits from `Transcript`. This data can be logged and later be used in case of disputes or complaints.

Identity file An identity file must be created for each actor in the applications. The identity file keeps configuration information such as the providers an actor wants to use in the framework and/or the location of credentials.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Identity>
  <Providers>
    <Provider>be.kuleuven.cs.idf.providers.secanon.Provider</Provider>
  </Providers>
  <StorageConfig repositoryPath="credRepository.xml">
    <StorageSettings id="1" name="XML" path="credDB.xml">
      <Technology name="XMLFile" version="0.0.1" />
    </StorageSettings>
  </StorageConfig>
</Identity>
```

Listing 3.4: Example identity file

The sample identity file keeps a pointer to one provider package. This provider supports eID, X.509 and Idemix credentials and TCP socket connections. The identity file also keeps configuration info about the location of existing credentials. The file `credRepository.xml` defines a list of `CredentialTemplate`s. This file is used by the `CredentialManager` to get an overview of the user's credentials without opening all credentials (which can be stored remotely). Credentials can be stored in an XML file, a Java keystore, a remote file, ... The parameters that are needed to open these storage locations, are defined in the `StorageSettings` tag. The example identity file uses an XML file `credDB.xml` to store the root credential of the issuer.

Issuer module The issuer listens for incoming connections, authenticates users and issues anonymous credentials. The source code below lists the source code of the issuer. Note that the framework is instantiated first (by creating a new `Identity` object that reads out the identity file).

```
Identity identity = new Identity(new File("issuer.idf"));
```

The issuer will keep a list of users that have already registered.

```
ArrayList<String> registeredUsers = new ArrayList<String>();
```

The issuer uses the CommunicationHandler to listen for incoming connections on TCP port 2001.

```
CommunicationSettings commSettings =
  new SocketCommunicationSettings("0.0.0.0", 2001);
CommunicationManager commManager = identity.getCommunicationManager();
CommunicationHandler commHandler =
  commManager.getCommunicationHandler(commSettings.getTechnology());
ConnectionListener listener = commHandler.startListen(commSettings);
while (true){
  Connection conn = commHandler.accept(listener);
```

If a client connects to the issuer, the issuer will require client authentication with the Belgian eID card. Therefore, he instantiates the corresponding `CredentialHandler` and calls the `receiveAuthentication` method. If the authentication fails, an exception is thrown. So, if the `receiveAuthentication` method returns a transcript object, the authentication was successful.

```
  Technology beid = new Technology("BelgianEid", new Version(0,0,1));
  ch = identity.getCredentialManager().getCredentialHandler(beid);
  AuthVerifierTranscript avt = ch.receiveAuthentication(conn, null);
```

At authentication, the issuer reads out the certain identifying data.

```
  String uniqueID = (String) avt.getReleasedAttributes().getValue("uniqueID");
  int age = (Integer) avt.getReleasedAttributes().getValue("age");
```

The issuer checks if that user already received an anonymous credential.

```
  if(registeredUsers.contains(nrn)) {
    commHandler.sendObject("Already registered");
  } else {
    commHandler.sendObject("OK");
```

Before the issuer can issue Idemix credentials, he has to load his own credential using the `StorageManager`.

```
    StorageManager sm = identity.getStorageManager();
    Credential issuerCred = sm.loadCredential(1);
```

An Idemix credential is issued by using the corresponding `CredentialHandler`. A credential template is loaded from XML. The attribute values are assigned.

```
    ch = identity.getCredentialManager().getCredentialHandler(idemix);
    Template template = new Template(new File("templateCredential.xml"));
    values = new AttributeValues();
    values.add(new AttributeValue("age", age));

    IssueTranscript it = ch.issueCredential(conn,template,values,issuerCred);
```

The user is added to the list of registered users in order to prevent that he can ask a another Idemix credential. At the end of the program, the issuer closes the connection.

```
    registeredUsers.add(uniqueID);
  }
  commHandler.disconnect(conn);
}
```

Ticket shop module The ticket shop module gives an overview of available tickets for all the events and handles purchases. The identity file of the ticket shop is first loaded and the server is listening for incoming connections on TCP port 2002.

```
Identity identity = new Identity(new File("ticketshop.idf"));
CommunicationSettings commSettings =
  new SocketCommunicationSettings("0.0.0.0", 2002);
CommunicationManager commManager = identity.getCommunicationManager();
CommunicationHandler commHandler =
  commManager.getCommunicationHandler(commSettings.getTechnology());
ConnectionListener listener = commHandler.startListen(commSettings);
while (true){
  Connection conn = commHandler.accept(listener);
```

The ticket shop module can receive two types of messages from the user, namely (1) a request for seat that are available for a specific event or (2) a request to buy some tickets. The seat list is included in an `Event` object in this example. The implementation of the `Event` object is not relevant for this example application.

```
  String message = (String) commHandler.receiveObject(conn);
  if(message.equals("freeseats")){
    ArrayList<Event> events = loadEventList();
    commHandler.sendObject(events);
  } else {
```

If the user wants to buy some tickets, he needs to sign the ticket numbers or seat numbers with his Idemix credential while proving that he is older than 18.

```
    Technology idemix = new Technology("Idemix", new Version(0,0,2));
    ch = identity.getCredentialManager().getCredentialHandler(idemix);
    Template userCredentialTemplate = new Template(new File("template.xml"));
    ShowSpecification spec = new ShowSpecification();
```

```
    spec.setAssertions(new BasicPredicate(userCredentialTemplate.getName(),
      "age", Operator.greaterOrEqual, 18));
    String message = (String) commHandler.receiveObject(conn);
    // check if message is valid and seats are still available
    SignVerifierTranscript svt =
      ch.verifySignature(conn, userCredentialTemplate, spec, message);
  }
  commHandler.disconnect(conn);
}
```

Buyer module A user first needs to register with his eID card and request an anonymous credential. With that credential, he must be able to buy tickets at the ticket shop. The first step in the implementation is loading the identity file and connecting to the issuer host.

```
Identity identity = new Identity(new File("buyer.idf"));
CommunicationSettings commSettings =
  new SocketCommunicationSettings("issuerHost", 2001);
CommunicationManager commManager = identity.getCommunicationManager();
CommunicationHandler commHandler =
  commManager.getCommunicationHandler(commSettings.getTechnology());
Connection conn = commHandler.connect(commSettings);
```

After the connection is established, the Belgian eID is used to authenticate to the issuer.

```
Technology beid = new Technology("BelgianEid", new Version(0,0,1));
ch = identity.getCredentialManager().getCredentialHandler(beid);
Credential cred = BelgianEidCredentialHandler.getCredentialObject();
AuthProverTranscript apt = ch.authenticate(conn, cred);
```

Thereafter, an Idemix credential is issued to the buyer.

```
Technology idemix = new Technology("Idemix", new Version(0,0,2));
ch = identity.getCredentialManager().getCredentialHandler(idemix);
Template template = new Template(new File("templateCredential.xml"));
GetCredentialResult result = ch.getCredential(conn, template, null);
Credential credential = result.getCredential();
// user should store his credential
```

The buyer closes the connection with the issuer and connects to the ticket shop.

```
commHandler.disconnect(conn);
commSettings = new SocketCommunicationSettings("ticketShopHost", 2002);
conn = commHandler.connect(commSettings);
```

The user can request the available seats for all events and select certain tickets. The user signs his ticket and/or seat numbers with his Idemix credential. It is important for Idemix that the `ShowSpecification` is the same at both the client and server side.

```
commHandler.sendObject("freeseats");
ArrayList<Event> events = (ArrayList<Event>) commHandler.receiveObject(conn);
// show list of free seats to user
String message = "Concert: 12-2, 12-4, 12-6" // string of selected seats
ShowSpecification spec = new ShowSpecification();
spec.setAssertions(new BasicPredicate(userCredentialTemplate.getName(),
  "age", Operator.greaterOrEqual, 18));
SignProverTranscript spt = ch.sign(conn, credential, spec, message);
commHandler.disconnect(conn);
```

3.4 Conclusion

This chapter presented a set of mobile technologies and advanced cryptographic building blocks that can be selected when building eID applications. An overview is given of each of these blocks. This overview may assist developers at the design phase. Developers need to select certain blocks depending on the level of security, usability and mobility that is required in a particular application. The cryptographic building blocks that are discussed can be used to realise an appropriate level of privacy. However, not all devices have substantial computation power to perform the necessary transactions in a reasonable time interval.
The identity framework offers design support in the sense that it allows to hide the complexity of the cryptographic building blocks. The design of the framework foresees that new technologies can be added using provider packages.

```
Data:
  Version: V3
   Subject: SERIALNUMBER=89021508512, GIVENNAME=Pieter Dirk,
      SURNAME=Verhaeghe, CN=Pieter Verhaeghe (Signature), C=BE
   Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5
   Key: Sun RSA public key, 1024 bits
   modulus: 96395658467377
   public exponent: 65537
   Validity: [From: Mon Jun 12 16:54:10 CEST 2006,
     To: Wed Jun 01 01:59:59 CEST 2011]
   Issuer: SERIALNUMBER=200509, CN=Citizen CA, C=BE
   SerialNumber: [10000000 000077e6 97bedc84 b2f8ecb4]
   Certificate Extensions: 7
    [1]: ObjectId: 2.16.840.1.113730.1.1 Criticality=false
      NetscapeCertType [S/MIME]
    [2]: ObjectId: 2.5.29.35 Criticality=false
      AuthorityKeyIdentifier [KeyIdentifier [
      0000: 4A 06 5E DA E1 F4 CD 7A B8 A5 1B A9 E2 AE DB F4 J.^....z........
      0010: 9A 7C 08 C3                                ....]]
    [3]: ObjectId: 2.5.29.31 Criticality=false
      CRLDistributionPoints [
       [DistributionPoint:[URIName: http://crl.eid.belgium.be/eidc200509.crl]]]
    [4]: ObjectId: 1.3.6.1.5.5.7.1.3 Criticality=false
      Extension unknown: DER encoded OCTET string =
      0000: 04 0C 30 0A 30 08 06 06 04 00 8E 46 01 01           ..0.0......F..
    [5]: ObjectId: 2.5.29.15 Criticality=true
      KeyUsage [Non_repudiation]
    [6]: ObjectId: 2.5.29.32 Criticality=false
      CertificatePolicies [[CertificatePolicyId: [2.16.56.1.1.1.2.1]
      [PolicyQualifierInfo: [qualifierID: 1.3.6.1.5.5.7.2.1 qualifier:
      0000: 16 20 68 74 74 70 3A 2F 2F 72 65 70 6F 73 69 74 . http://reposit
      0010: 6F 72 79 2E 65 69 64 2E 62 65 6C 67 69 75 6D 2E ory.eid.belgium.
      0020: 62 65                                     be]
    [7]: ObjectId: 1.3.6.1.5.5.7.1.1 Criticality=false
      AuthorityInfoAccess [[accessMethod: 1.3.6.1.5.5.7.48.2
       accessLocation: URIName: http://certs.eid.belgium.be/belgiumrs.crt,
       accessMethod: 1.3.6.1.5.5.7.48.1
       accessLocation: URIName: http://ocsp.eid.belgium.be]]]
    Algorithm: [SHA1withRSA]
    Signature:
     0000: BA E2 49 ED 33 28 24 1B DE 8B EC 22 16 D7 46 E0 ..I.3($....".."..F.
     ...
     00F0: 1B F6 33 24 1F 92 BA C4 16 D1 44 D7 44 A0 D4 82 ..3$......D.D...
```

Figure 3.2: Example of a X.509 certificate of used in the Belgian eID card

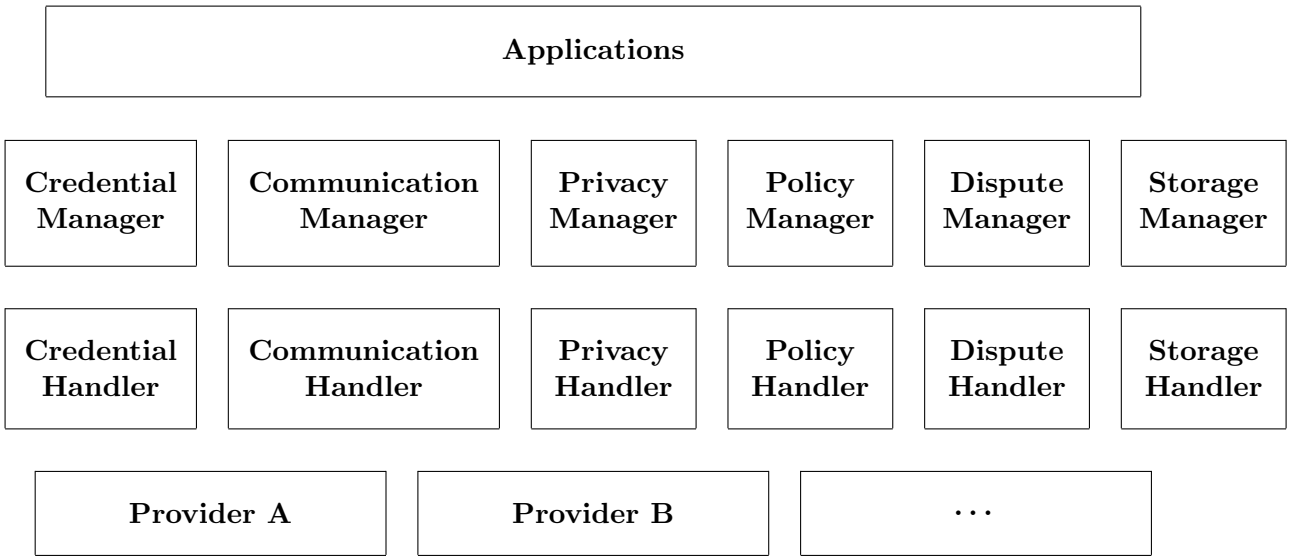


Figure 3.3: Overview of framework components

Chapter 4

Applications

Multiple proof-of-concepts are developed in the scope of the eIDea project. Three of them are discussed below. First, a domotic application is presented. The prototype demonstrates how the Belgian eID can be used as a strong authentication mechanism for remote control of the home automation system through the Internet. Second, a privacy friendly ticketing system was designed. Two alternatives are presented. The eID card is used to retrieve an anonymized permit, allowing a user to order tickets anonymously. A first solution is based on pseudonym certificates, i.e. X.509 certificates containing a user's nym instead of a real identity. A second solution is based on the more enhanced anonymous credential systems. The latter allows to anonymously disclose only a subset of the personal attributes (or properties thereof) embedded in the credential. A prototype is implemented based on pseudonym certificates. Finally, the ePoll application is a privacy-preserving opinion poll system using anonymous credentials. It combines reasonable privacy properties with reliable results. The prototypes of the ticketing application and the ePoll system are built on top of the identity framework that is discussed in the previous chapter.

4.1 Access control

4.1.1 Introduction

Home automation provides the automation of different tasks in private homes to increase comfort, security and intelligence of the system. Besides building automation, like controlling lights, it includes multi-media functionality, automation of recurring patterns, alarm functions, etc. The ultimate goal of a home automation system is to make life more convenient.
Traditional home automation systems typically consist of multiple hardware modules (i.e, *inputs* and *outputs*) that are controlled by a dedicated hardware *controller*. The controller receives input signals from the input modules and reacts by sending outputs to the output modules. For instance, a controller may detect that someone pressed a light switch. The controller will react by sending output signals to certain lights.
Some researchers propose new hardware models to enhance functionality in home automation [26, 59, 55, 37]. However, baking the control into a hardware controller has several disadvantages. First, this approach is not flexible. Modifying functionality in the home automation system is very hard to achieve. Second, the tasks of a controller may be very complex. The output signals that are sent may depend on certain context. For instance, a door may only be opened if a person authenticated within a certain time interval. It may even be required that

certain output signals are delayed. For example, the house owner may want to program the heating. Third, many home automation systems do not offer appropriate support to modify the state of the home automation system remotely [36, 45]. However, enabling remote control raises additional security requirements [54].
The contribution of this section is twofold. First, it presents a *generic* and *modular* architecture for building home automation servers. A home automation server typically enables controlling, managing, modifying and monitoring of one or more home automation (sub)systems. Hence, the architecture can be used to extend many home automation systems with a software server. Moreover, the layered approach enables the reuse of a particular server for alternative (sub)systems [29]. Modularity aims at increasing flexibility and configurability. Second, the server can be accessed through an IP network. The latter enables *remote control* through the Internet. The prototype foresees a strong authentication mechanism (based on the Belgian Electronic Identity Card) to contribute to security measures.
The remainder of this section is structured as follows. The design of the home automation server is presented in section 4.1.2. The model is further extended with multiple modules to enforce security requirements. Section 4.1.3 describes the secure home automation server. Section 4.1.4 evaluates the system and compares it to other related systems. This section ends with general conclusions and future work.

4.1.2 Home automation

This section presents a generic architecture for building home automation servers. Hence, control decisions can be moved from the physical controller to the software server. To perform its tasks, the server regularly polls the state of the home automation system (i.e, the states of all input modules and output modules) and translates the current physical state into a logical state. It subsequently checks if the logical state should result in a modification of certain output states. If so, the server translates the logical output to a physical output and sends it to the physical controller that, on its turn, modifies the state of certain physical components. The next paragraph discusses the layered architecture in more detail.

Physical view versus logical view

As already outlined before, the home automation server maintains two views on the system, namely a *physical view* and a *logical view*. The physical view represents a model of all hardware components (i.e, inputs, outputs, controllers, states, ...) in the system. The logical view redefines certain concepts at a higher level of abstraction. For instance, a light switch is a logical unit that may consist of multiple physical modules. Each physical module is a unit of granularity at the physical level. The light switch is a unit of granularity at the logical level. Administrators typically want to manipulate logical modules instead of physical atomic modules. Maintaining multiple views on the system has two main advantages. The physical view enables to map configurations to a particular system. The logical view eases configuration and control of the system by administrators.

The physical view. In the physical view, a *Home Automation System (HAS)* consists of one or more subsystems. Each *subsystem* consists of a controller and multiple modules. These modules are connected (through a bus) and are controlled by the controller. Each *module* has a unique address and holds the technical details of the module. Moreover, each module consists

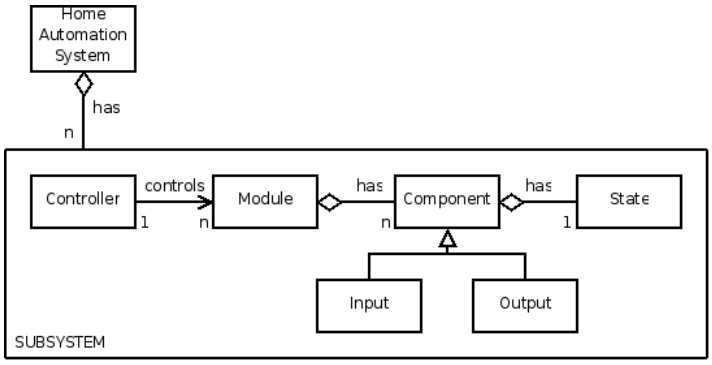


Figure 4.1: Overview of the physical view

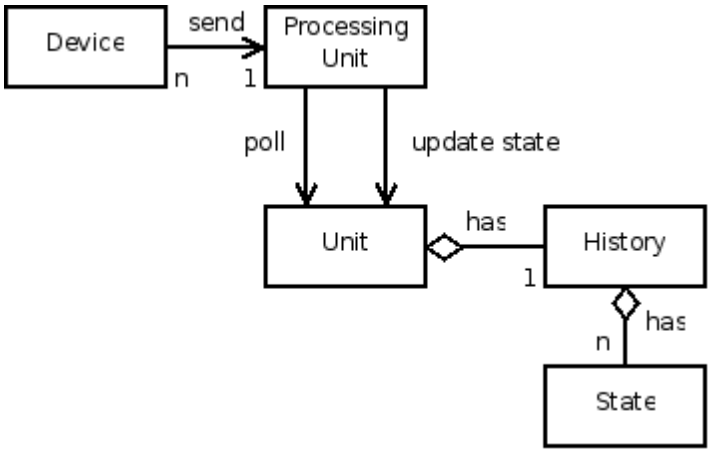


Figure 4.2: Overview of the logical view

of one or more components. A *component* is either an input or output. Each component has a *state*. Figure 4.1 gives an overview of the physical view. The communication is initiated by the controller. To perform this task, it uses the addresses of the different modules to poll the state of their inputs and outputs. State information is used to make decisions. For example, a light can be turned on if the state of a certain switch has changed. Note that the actual control decisions are delegated to the processing unit in the logical view.

The logical view. The logical view defines the Home Automation System at a higher level of abstraction. Among others, it abstracts the technical details of hardware modules, the decomposition of the Home Automation system in subsystems, ... The logical view is depicted in figure 4.2. Each *unit* represents a logical component such as a light switch, a keyboard to enter a pin-code, a series of lights that logically fit together, ... Moreover, each unit keeps a history. The *history* defines the current state and the set of previous states. The *processing unit* is the core of the home automation server. It performs multiple tasks. First, it polls the logical units at regular time intervals and stores the state of each unit. Second, the processing unit is responsible for changing the state of logical units. Multiple

devices can be connected to the processing unit. These devices are often complex systems (e.g. mobile devices, remote computers, ...) that cannot connect directly to the bus system. The link between physical and logical view is demonstrated in figure 4.3. The decision can be based upon a set of rules that are evaluated at regular time intervals. The next paragraph elaborates on a formal control language.

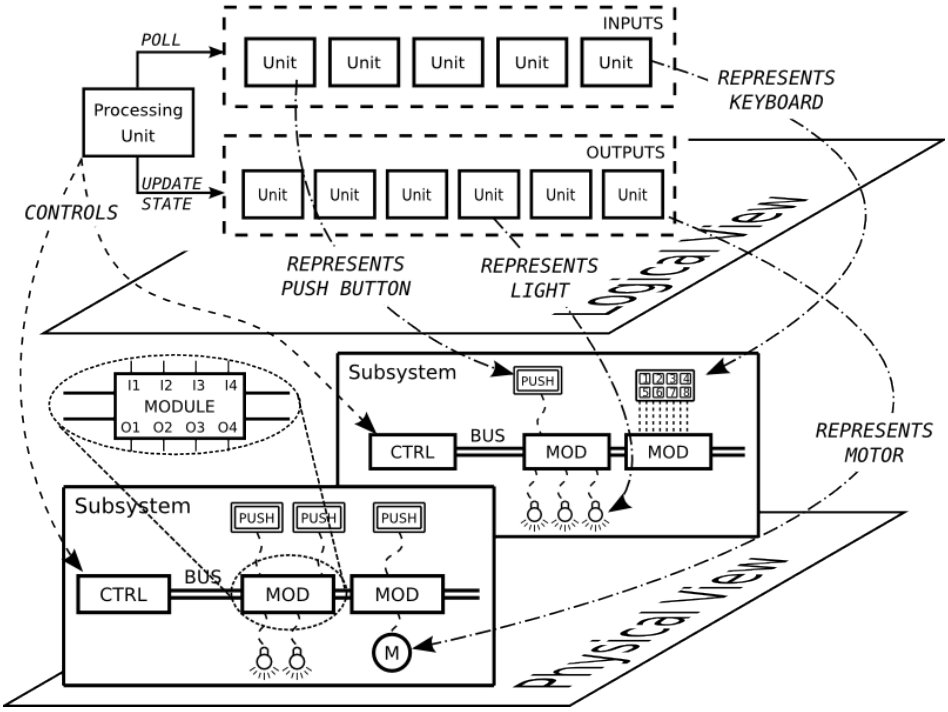


Figure 4.3: Link between logical and physical view

A formal control language

The processing unit can update the state of the logical units based on the current and/or previous states of (other) units. This section introduces a formal language [29, 34] to express control decisions. Each U_x defines a logical unit. S_{U_x} defines a subset of states of unit U_x. T_y defines a time interval. Each rule consists of a left hand side and a right hand side. Both sides consist of a set of three-tuples (U_x, S_{U_x}, T_x). Hence, each rule is defined as follows:

$(U_1, S_{U_1}, T_1), \ldots, (U_n, S_{U_n}, T_n) \rightarrow (U'_1, S_{U'_1}, T'_1), \ldots, (U'_m, S_{U'_m}, T'_m)$

The expressive power of our language is illustrated with two examples. The first rule defines that the heating must be activated directly if the temperature in the building is lower than 18 degrees between 18h and 24h.

$('temperature\_sensor', [-\infty, 18], [18h - 24h]) \rightarrow ('heating', 'on', now)$

The second example defines that if the light sensor detects that it is dark and if the motion detector detects that someone enters the house, the following actions must take place: $light_1$ must be activated immediately and $light_2$ must be activated in 3 seconds.

$$('light\_sensor','dark',now),('motion\_detector','move',now) \\ \rightarrow ('light\_1','on',now),('light\_2','on',now+3s)$$

Discussion

This section evaluates the server architecture on multiple parameters, namely interoperability, simplicity and usability.

Interoperability. The lack of clear standards in the home automation industry [53] results in the coexistence of many non-compatible solutions in the market. Nikobus [13] and Contatto [5] are two of them. Industrial manufacturers do not have clear insights on the most suitable technology for their products and their future compatibility with other domestic appliances and products. The home automation architecture discussed above, abstracts the details of these hardware issues and allows users to configure the house automation system without getting lost in the mires of the implementation.

Simplicity and usability. The two views in the model hide the details of the hardware for the end user. A device is modelled as a simple input or output, no matter how it is implemented. A switch is, for instance, an input with states on and off, while in another case it is a temperature sensor with a range from 0° to 30° Celsius. A user can monitor and use this input, without having to wonder how this state is actually extracted from the input device. Another advantage is that a more simple controller can be used as a relay to the server. The control algorithms do no longer have to be submitted to the controller, but remain on the server. This enables support of more complex scenarios than in traditional home automation systems.

To demonstrate the feasibility of our approach, we applied the proposed architecture to the development of a software server for the Contatto HAS from Duemmegi[5].

4.1.3 Secure Home Automation

Many scenarios can be found where remote control may be desired [27]. The owner of the system may want to start the micro wave or control the temperature in his house remotely. Or the owner may want to enable other persons to enter the house temporarily. However, enabling remote control implies appropriate security measures. First, users can only change the state of the Building Automation System after they authenticated successfully. Moreover, logging may help to detect misbehaviour and to enable appropriate control measures. This section gives an overview of the extended architecture.

Home Automation Server and Management Server. The Secure Home Automation system consists of two servers, namely a *home automation server* and a *management server*. The purposes and functionalities of the *home automation server* are described in the previous section. The *management server* enables the reconfiguration of the system and the restriction of remote access to services. Privileged entities (typically the owner of the system and maybe some additional persons) can use the management server for three purposes:

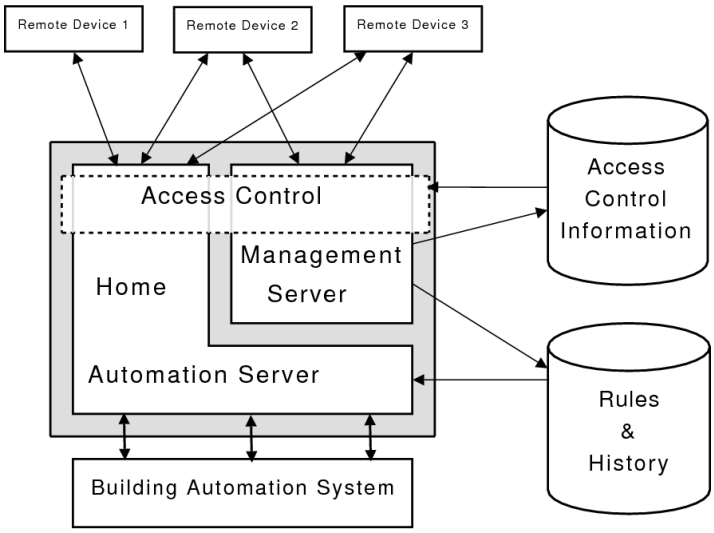


Figure 4.4: Overview of the Secure Home Automation System

- privileged users can (re)configure the types of actions that must be performed if the state of one or more input modules has changed. For instance, pressing a switch will have a different outcome after reconfiguration.

- privileged users (typically the owner of the building or house) can add/remove users and assign/change roles to these users. For instance, the owner of the building can add his/her partner as an additional administrator and assign roles with more restricted privileges to the children.

- privileged users can define an access policy and assign that policy to certain roles. An access policy typically consists of a set of rules or permissions that define the types of actions that can be performed by users within that role.

The first functionality aims at easing reconfigurability of the home automation system. The latter functionalities aim at supporting secure remote access. Note that a policy is not assigned directly to a person. A policy is assigned indirectly to a person through the concept of a role [42]. This enables a more flexible and consistent access management. If a new user is added, one or more roles have to be assigned to that user. We do not need to define a set of permissions for that user explicitly. Changing a policy has an impact on all users within a particular role. The latter aims at defining a consistent management.

Access control module. Remote access to the home automation system has to be restricted. Only privileged users may (de)activate *outputs*. Similarly, access to the management server has to be restricted. Only a few users may change policies and/or add/remove users. During the security analysis, we assumed that both the management server and the actual home automation server can be accessed over an insecure communication channel.
Adversaries can cause a lot of damage in recent home automation systems. Whereas home automation systems were initially used to control only heating and/or lights, current systems

aim at easing control of more critical functions (such as micro wave, door locks, ...). Consequently, strong authentication mechanisms are advisory. If a user initiates a connection to either server from a remote machine, a secure connection (i.e, an SSL connection) is established between both machines. Mutual authentication is required. The servers authenticate to the remote hosts by means of a server certificate that is embodied in the server software and is typically issued by a trusted third party. Users can authenticate to a server by means of multiple types of authentication tokens. One alternative is to support password-based user authentication [32]. However, password-based authentication has multiple disadvantages. First, passwords are difficult to administer. Second, passwords can easily be delegated to possibly *untrusted* users. The prototype that is developed supports user authentication by means of the Belgian electronic identity card. Even if an adversary succeeds to steal a valid identity card, that token is useless as a pin code is required to go through the authentication phase successfully.
After a user authenticated successfully (i.e, the user was added to the system previously), that user can perform actions in the system. At each action, the server verifies if the user has the permission to perform that action. Hence, some users may not unlock a door remotely whereas more privileged users may have the appropriate roles (and permissions) to unlock that door remotely.
Note that the state of the system may change because of two reasons. First, the state of output modules can change because of state changes of internal input modules. Examples are given in section 4.1.2. Second, the state of output modules can change because of a remote action. However, remote actions require an appropriate set of permissions.

4.1.4 Evaluation

The proposed formal control language enables to define rules to configure the domotics system. Using this simple formal language, many configurations can be expressed. However small modifications can make things easier. Adding logic operators increases usability, since multiple rules can be expressed into one. For instance, 'if *button_1* or *button_2* is pressed, *light_1* is switched on' can be expressed as follows:

$('button\_1', on, now) \rightarrow ('light\_1',' on', now)$
$('button\_2', on, now) \rightarrow ('light\_1',' on', now)$

become

$('button\_1', on, now) \vee ('button\_2', on, now) \rightarrow ('light\_1',' on', now)$

With role-based access control, access decisions are based on roles that individual users have within an organization [50]. Permissions are not assigned directly to users, but users acquire them through roles, simplifying administration and management of privileges. Roles can be updated without updating privileges for every user individually. However, to make a minor change in permissions for an individual, a new role has to be created. Therefore it might be appropriate to add permissions directly to the user. Moreover simple role-based access might have other disadvantages. Unlike context based access control [44], role-based access control does not consider message context such as the location of the initiator of a connection. In home automation systems, access from untrusted networks requires more constraints than

access from the local home network. An administrator, for instance, should not modify low level settings when connecting through the Internet. Hence, damage caused by malicious adversaries on untrusted networks is kept to a minimum. Future research will address these shortcomings in the prototype of the Secure Home Automation System.

The prototype supports remote authentication based on the Belgian electronic identity card [28]. Although the card provides a powerful means for strong authentication, using the card may also have some disadvantages. First, only Belgian citizens can authenticate. Until now, about 70 percent of the Belgian citizens possess an electronic identity card. However, we assume that within a few years, each Belgian citizen has one. Second, the user must be on a remote location where a machine with card reader is available to authenticate to the system. Future research will address strong authentication mechanisms so that the servers are accessible from mobile devices such as a GSM and/or PDA. The SIM card in a GSM contains a unique identifier that also can be exploited to authenticate to the Home Automation System.

4.2 eTicketing

4.2.1 Introduction

Tickets are used for an innumerable number of events: soccer matches, music festivals, exhibitions, etc. These tickets are ever more bought electronically. An increasing number of countries issue electronic identity cards to their citizens. Examples are Belgium, Estonia and Austria. These eID cards usually allow the holder to authenticate and to digitally sign documents, but often, they are very privacy unfriendly. For example, authentication using the Belgian eID card will usually lead to the divulgement of important personal data such as your national registration number (NRN). Despite these privacy dangers, the use of the eID card is promoted by the governments. We can thus expect that in the near future, electronic ticketing systems will arise based on the eID card. A trivial solution is easy to devise. However, this solution is not acceptable because it further endangers the card holder's privacy as profiles can easily be compiled, linked to each other and to the identity of the card holder. An advantage of the use of eID cards is that it is straightforward to impose restrictions on the maximum number of tickets that can be bought by one user, hence, thwarting the sales on black markets. Sometimes, special offers are available for buyers under or over a certain age or living in the region where the event is organized. Here too, eID cards can help in securely conveying (proving) that these conditions are satisfied for the buyer. However, the use of these cards will usually disclose more information than is required.

For big events with thousands of attendants, the police would be helped if tickets were not anonymous, but could be linked to the identity of the attendants, or at least to the identity of the buyers of these tickets. Especially, when riots occur, it would make it easier to identify and prosecute the instigators. However, the use of tickets attributable to individuals poses severe privacy risks and brings us closer to a "Big Brother" state.

This section proposes two solutions where the eID card is needed to obtain an anonymized permit, allowing a user to obtain tickets in a privacy friendly way. The role of the eID card is thus reduced to a bootstrapping role. A first solution is based on pseudonym certificates, i.e. X.509 certificates containing a user's nym instead of a real identity. A second solution is based on the more enhanced anonymous credential systems, which allow to anonymously disclose only a subset of the personal attributes (or properties thereof) embedded in the credential.

Both solutions are validated and compared with the trivial solution and with each other.

The main requirements are given in section 4.2.2. Section 4.2.3 explains notations and specifies the assumptions. Sections 4.2.4, 4.2.5 and 4.2.6, discuss the trivial protocol and two privacy friendly alternatives and are followed by a comparison in section 4.2.7.

4.2.2 Requirements

The protocols that are discussed will be evaluated based on the following requirements. F4 and F5 are optional.

Functional/Security Requirements

F1 Every event may have a policy that limits the number of tickets obtainable by one buyer. The limit may depend on a property of the buyer.

F2 Event organizers may choose to offer a subscription for a series of events.

F3 Every event can have a pricing policy that differentiates between different groups of buyers (e.g. youngsters or elderly people).

(F4) When abuse is detected or when serious incidents happen during the event, it should be possible to identify the buyer of the ticket(s) involved.

(F5) Individuals who have been imposed a banning order for a particular event type, should not be able to buy tickets for this kind of events.

Privacy Requirements

P1 Buyers of tickets should not directly be identifiable.

P2 Except when subscriptions are used, it should not be possible to compile buyer's profiles.

P3 It should not be possible to identify an individual on a blacklist.

4.2.3 Assumptions and Notation

The general assumptions and notation w.r.t. the protocols are now summed up.

Assumptions

- For every protocol, a server always first authenticates to U using a classical X.509 certificate. Also, an integrity and confidentiality preserving connection is established during a protocol. Anonymity at the network layer is added when necessary.
- A ticketing server can service multiple events. However, for each event, there is only one ticketing server.
- Tickets do only contain a ticket identifier (e.g. event name, date and seat number) and are unforgeable.

Notation

- Each protocol requires the following roles: user U (client), ticket server T (issues tickets to users), event organizer E and the court of justice J.
- $U \leftrightarrows B \leftrightarrows T$: $(\text{PayProof}_\text{U}, \text{PayProof}_\text{T}) \leftarrow \mathsf{pay}(price, Msg)$. U pays an amount of money, via an intermediary bank B, to T. A message can be linked to the payment. The bank can deliver proofs of the payment to both parties. The payment protocols can preserve U's privacy.
- $U \leftrightarrows T$: $(desc[], price, [Proof]) \leftarrow \mathsf{negotiate}(cert \vee Cred, Nym \vee Id, event, eventPolicy, \#tickets, specification)$ allows U and T to agree on the exact seat numbers as well as on the total price. Therefore, U gives an identifier (Nym or Id), shows (properties of) credential/certificate attributes. The event policy can state e.g. that people younger than 18 get reductions. Evidently, the number and (general) specification of the tickets are given as well. The restrictions on the blacklists can further constrain the possibilities of the user. U can give T a proof of the agreement (signed by $cert$ or $Cred$).
- O: $\text{Nym} \leftarrow \mathsf{retrieveOrGenerateNym}(Id \vee \text{Nym}')$ returns a newly generated nym if the user referred to by Id or Nym' does not yet have a nym with O. However, if that user already has been given a nym in the past, it is simply retrieved from the local storage system.
- T: $Restrictions \leftarrow \mathsf{retrieveRestrictions}(Blacklist, Nym \vee Id)$. The ticketing service looks up the restrictions of a Nym or Id in a blacklist.
- G: $\text{Restriction}[] \leftarrow \mathsf{getRestrictionBooleans}(Id)$ implicitly uses all blacklists, and returns for each event type whether or not the user is blacklisted or not.
- Other, self explaining methods are: $\mathsf{add}()$, $\mathsf{lookup}()$, $\mathsf{store}()$, update () and $\mathsf{generateTickets}()$.

4.2.4 Trivial eID-based Solution

Introduction

Without alternatives, this protocol will most likely be be implemented in Belgium as the government is really promoting the use of the eID card in commercial applications. However, this protocol has serious privacy drawbacks.

High Level Description

U uses his eID card to authenticate to T, revealing a lot of personal data to T. A government agency G maintains a blacklist containing identifiable user ids. This blacklist is checked by T before issuing tickets.

Protocols

The protocols are given in table 4.1 and are quite self explaining. The user authenticates to T using his eID card. T first checks whether the user is blacklisted. Based on the user's id and personal attributes, the user can be given the possibility to buy a number of tickets as a

result of the negotiation phase. After the payment and ticket issuance, T finally stores ticket selling info.

Dispute handling is straight forward: since T knows the *link* between the seat (or ticket) and the user's id.

(4.1.a) Getting a ticket

(1)	U → T	:	authenticate(*eID*)
(2)	T	:	*Restrictions*← getRestriction(*eID.NRN*, Blacklist[EventType])
(3)	U ⇆ T	:	(SeatNB[], Price) ← negotiate(*eID*, eventPolicy, Restriction[])
(4)	U, T	:	if (SeatNb[] = ∅) abort
(5)	U ⇆ B ⇆ T	:	pay(price, H(SeatNR[], ...))
(6)	U ← T	:	tickets[] ← generateTickets(SeatNb[])
(7)	T	:	update [*eID.NRN*, event, tickets[]]

(4.1.b) Maintaining the blacklists

(1)	J → G	:	*eID.NRN*, Restrictions, eventType
(2)	G	:	Blacklists[EventType].add(*eID.NRN*, Restrictions)
(3)	G → T	:	Blacklists

Table 4.1: Protocols in trivial implementation

Evaluation

The functional/security requirements are trivially fulfilled. However for the privacy requirements, this protocol fails completely. T knows the user's id and all other attributes contained in the eID certificate (P1). User profiling is trivial for T as well as sharing and linking of profiles (P2). The users' ids are on the blacklist (P3). In addition, many countries simply forbid blacklists on which users are identifiable due to privacy legislation. Deployment will often thus result in omitting the F5 requirement.

4.2.5 Solution based on Pseudonym Certificates

Introduction

This approach improves the privacy of the user by using pseudonymous permits. A unique pseudonymous root certificate is issued by the government. This allows the user to obtain pseudonymous permit certificates from different permit servers. One permit server could for instance be responsible for one event type (e.g. soccer matches). With such a pseudonymous permit a user can buy tickets for events that happen in a small (permit specific) time period[1]. The user will thus most likely need multiple permits. The blacklists no longer contain user identifiers, but pseudonyms.

Roles

Besides the already defined U, T and E, a government agency G is needed to issue root certificates and a permit server PS issues permit certificates.

[1] The fixed time period is introduced to minimize linkabilities.

Assumptions

- All certificates contain a unique serial number, a pseudonym or id, a public key and an expiry date.
- There are many pseudonym servers (PS) and many ticket servers (T).
- For every event, the ticket server (T) accepts permits. issued by a limited set of pseudonym servers. However, the user sets of different pseudonym servers do not overlap (necessary for requirement F1).
- Only one entity G can issue valid pseudonymous root certificates.
- Nyms that are no longer valid, are forgotten by the permit server.

High Level Description and data structures. The user receives a pseudonymous root certificate (Cert^{R}), which contains a rootnym (Nym^{R}) and possibly other attributes (such as year of birth, citizenship, place of residency, ...). Cert^{R} is used to authenticate to the permit server PS.
The user can apply to the PS for a pseudonym (Nym^{P}) that is valid during a predefined time period. Nym^{P} will be certified in a (pseudonymous) permit certificate (Cert^{P}). Each certificate also contains a public key used to verify authentications and signatures with Cert^{P}, and possibly (properties of) other attributes that were copied from the root certificate (Cert^{R}). Using permit certificates with non-overlapping time-slots, each user can have at most one valid Cert^{P} to order tickets for a particular event. The PS can refuse permits to users who have been sentenced to a banning order for events supported by the PS.

Protocols

Getting a root certificate. A governmental instance G issues a single pseudonymous root certificate Cert^{R} to each citizen. This Cert^{R} contains a pseudonym Nym^{R} that was either newly generated or retrieved by G in case the user was already assigned a Nym^{R} in the past. The user can request G to copy (properties of) attributes from in his *eID* card into his Cert^{R}[2]. G finally stores the user's *NRN* and Cert^{R}s (which include Nym^{R}.)

Getting a permit certificate. U authenticates with a valid root certificate Cert^{R} to the PS. PS will issue a number of permit certificates Cert^{P}s which have to be used before a (user specified) date (validThru). For instance, the user can request permit certificates that allow him to buy soccer tickets for the upcoming year. PS generates a set of nyms (Nym^{R}) or retrieves them (if they were already assigned in the past): one nym per time period[3]. Each nym Nym^{P} is also certified in a permit certificate Cert^{P} which also contains a validity period (for Nym^{P}), possibly a set of attributes and an encryption of the user's root pseudonym Nym^{R}. The validity periods of Nym^{P}s are non-overlapping. Hence, users cannot buy tickets for the same event using different nyms. Also, when a user requests a new permit for the

[2] The user can request several root certificates, each including a different set of properties/attributes. However, all certificates will refer to the same Nym^{R}.

[3] The length of the non-overlapping time periods is chosen by the PS in such a way that the number of events that fall in each period is limited.

same period (e.g. because the previous one was lost or the private key was stolen), PS will always use the same nym (Nym^{P}). Each Cert^{P} contains a probabilistic encryption of Nym^{R} with the public key of PS. This allows to enforce certain control measures in case of abusive behaviour (see further). PS finally updates the list of Cert^{P}s that are issued to Nym^{R}. PS can derive the time intervals for which a Nym^{R} has obtained a valid Cert^{P} from that list.

Buying tickets for an event. The user first authenticates to the ticket server T using the permit certificate Cert^{P} that is valid for that specific event and specifies the number of tickets he wants to order. T then verifies whether the pseudonym Nym^{P} is blacklisted for that event. T also checks whether Nym^{P} can still order the requested number of tickets for that event. If both conditions are fulfilled, the user and the ticket server agree on the price of the tickets and the seats. The price can depend on certain attributes that are embedded in the permit certificate (such as the user's age). After payment, the user retrieves the tickets. Finally, the ticket server updates the number of tickets that are sold to Nym^{P} for that event.

Updating anonymous blacklists. To fulfil requirement F4, anonymous blacklists are used. Four entities are involved in updating blacklists (see table 4.3).
A law enforcement entity J forwards the court orders (NRN, banning order) to G. G substitutes the NRNs with the corresponding Nym^{R}s and forwards the list to the permit server PS. PS can then add Nym^{R} to a blacklist for certain event types (i.e. PS will no longer issue Cert^{P}s to Nym^{R} for the event types that are specified in the blacklist).
Finally, PS retrieves the valid Nym^{P}s for each Nym^{R} with a banning order, substitutes every Nym^{R}-record in the blacklist with a number of Nym^{P}-records and forwards the new list to the ticket server T. T no longer issues tickets to pseudonyms in the blacklist. Note that the ticket service can even revoke tickets that were already issued to pseudonyms in the blacklist.

Identifying buyer of a ticket To reveal the identity of a participant with a specified seat number, the ticket service T looks up the Nym^{P} of the user that ordered the ticket. The corresponding permit certificate Cert^{P} is kept by the ticket server and is passed to pseudonym server. The latter can link Cert^{P} to Nym^{R} (as Nym^{R} is encrypted with the public key of the pseudonym server in Cert^{P}). G can reveal the user behind Nym^{R} (as G knows the mapping between NRN and Nym^{R}). Note that a law enforcement entity J typically intermediates in the deanonymization procedure (see table 4.3).

Evaluation

F1 This requirement is easily fulfilled as each user has only one $\text{Nym}^{\text{P}}$to buy tickets for a particular event.

F2 If Nym^{P} can be used to order tickets for multiple events (such as multiple soccer games during a World Cup contest), T can even restrict the total number of tickets that can be bought for the whole contest (i.e. a set of events).

F3 A user can get a lower price for some tickets based on the attribute values of Cert^{P}. However, tickets can be passed on. Hence, T should be careful with price reductions.

F4 Fulfilled (cfr. *"Identifying buyer of a ticket"* protocol).

(4.2.a) Getting a pseudonymous root certificate Cert^{R}

(1)	U → G	:	authenticate(eID)
(2)	G	:	Nym^{R}← retrieveOrGenerateNym($eID.NRN$)
(3)	U ← G	:	Cert^{R}← issueCert({Nym^{R}, attributes ...})
(4)	G	:	store[$eID.NRN$, Cert^{R}]

(4.2.b) Getting a permit certificate Cert^{P}

(1)	U → PS	:	authenticate(Cert^{R})
(2)	U → PS	:	validThru, *attributes to include*
(3)	PS	:	∀ [from,till], from ≤ validThru:
(4)	PS	:	Nym^{P}← retrieveOrGenerateNym(Cert^{R}.Nym^{R}, [from,till])
(5)	U ← PS	:	Cert^{P}← issueCert(Nym^{P}, [from,till], *attributes* $\text{enc}_{pk_{SP}}$(RAND || Cert^{R}.Nym^{R}),)
(6)	PS	:	store[Cert^{R}.Nym^{R}. [from,till], Cert^{P}]

(4.2.c) Buying tickets

(1)	U → T	:	authenticate(Cert^{P})
(2)	U → T	:	event, #tickets, specification
(3)	T	:	*Restrictions*← retrieveRestrictions(Cert^{P}.Nym^{P}, EventType)
(4)	U ⇆ T	:	(SeatNb[], price) ← negotiate(Cert^{P}.Nym^{P}, event, #tickets, eventPolicy, Cert^{P}.attr, specification, [*Restrictions*])
(5)	U, T	:	if (SeatNb[] = ∅) abort
(6)	U ⇆ B ⇆ T	:	pay(price, Hash(SeatNb[], ...))
(7)	U ← T	:	tickets[] ← generateTickets(SeatNb[])
(8)	T	:	update [Cert^{P}, event, tickets[]]

Table 4.2: Protocols with pseudonym certificates

F5 Three entities are needed to ban a user from event types for which a user already has a permit certificate, namely G, PS and T. Two entities are needed to ban a user from event types for which a user does not yet have a permit certificate, namely G and PS.

P1 As discussed in *"Identifying buyer of a ticket"*, four entities are needed to reveal the user's identity. Moreover, G (and maybe PS) are governmental instances. Hence, users can trust that players in the commercial sector (such as E and T) cannot identify users without help of governmental instances.

P2 Each Nym^{P} only has a limited validity period. The number of tickets that is issued to the same Nym^{P} is restricted. Hence, T and E can only compile limited profiles. PS can link all Nym^{P}s to the same Nym^{R}. However, multiple pseudonym servers PS can be used. If each PS can only issue permit certificates for specific types of events, the one PS cannot link multiple interests of the same Nym^{R}.

P3 Only Nym^{R}s and Nym^{P}s are kept in blacklists.

(4.3.a) anonymizing the blacklists

(1)	J → G	:	[*NRN*, banning order eventType]
(2)	G	:	$\text{Nym}^{\text{R}} \leftarrow \mathsf{lookupNym}(NRN)$
(3)	G → PS	:	[Nym^{R}, banning order eventType]
(4)	PS	:	$\text{Nym}^{\text{P}} \leftarrow \mathsf{lookupNym}(\text{Nym}^{\text{R}}, \text{eventType})$
(5)	PS → T	:	[Nym^{P}, banning order eventType]

(4.3.b) Identifying buyer of a ticket

(1)	J ← E	:	complaint, seatNb
(2)	J → T	:	event, seatNb
(3)	J ← T	:	$[\text{Cert}^{\text{P}}$, event, ticket$] \leftarrow \mathsf{lookup}$(event, seatNb)
(4)	J → PS	:	Cert^{P}
(5)	J ← PS	:	$(\text{RAND} \mid\mid \text{Nym}^{\text{R}}) \leftarrow \mathsf{dec}_{prk_{SP}}(\text{Cert}^{\text{P}}.\text{enc})$
(6)	J → G	:	Nym^{R}
(7)	J ← G	:	$NRN \leftarrow \mathsf{lookup}(\text{Nym}^{\text{R}})$

Table 4.3: Protocols with pseudonym certificates (bis)

4.2.6 A Ticketing System Based on Anonymous Credentials

Introduction

We further increase the user's privacy. The user needs a single permit - issued by a government agency - which allows the user to buy tickets for every event. In case of abuse, the transcript resulting from the permit show can be deanonymized. For each event type, there is a privacy-preserving blacklist, summing up the user's rights restrictions.

Roles

Besides U, E, T, and J, we define G as a government agency that issues permits and manages blacklists.

Assumptions

In the ticketing system based on anonymous credentials, we assume the following:

- The anonymous credential system provides the unlinkability property to permits. The user does not reveal identifiable permit attribute properties.
- All Es and all Ts and G have a unique, publicly available provable one-way function; $\text{f}^{\text{E}}()$ for E, $\text{f}^{\text{T}}()$ for T) and $\text{f}^{\text{G}}(.\,,.)$ for G. Note that the latter requires two arguments. These functions could for instance be included in their X.509 certificate.
- The opening info generated by a commit method does not reveal any information about the content contained in the commitment. This is easily achieved using a symmetric key K:
  $Com^{new} \leftarrow (Com,\ enc_K(\text{OpenInfo}))$ $\text{OpenInfo}^{new} \leftarrow K$ combined with integrity preserving measures (e.g. MACs).

High Level Description

The permit is an anonymous credential containing a set of personal attributes, a boolean value for each event type indicating whether or not the user is blacklisted, and two nyms. One nym (Nym^{R}) is known by G and used to blacklist persons. The other nym (Nym^{P}), is not known to G, but is used to generate an event specific nym, allowing T to keep track of the number of tickets sold to that person for that specific event.

Per event type, a blacklist is maintained by G. This blacklists contains user pseudonyms (Nym^{R}s). These nyms are converted to event specific nyms (Nym^{E}s) before the blacklist is sent to a specific T as a way to avoid linkabilities.

Protocols

Getting an anonymous Permit Certificate. The actual issue of the permit (4.7.a.5) includes a subset of the user's personal attributes (*attributes*) contained in the user's eID. These can be selectively disclosed during a credential show protocol.

The permit contains for each event type a boolean *Restrictions*[EventType] stating whether or not the user is blacklisted. G can easily extract this information out of the blacklists it manages (cfr. below).

Each permit contains two user unique pseudonyms Nym^{R} and Nym^{P}. Nym^{R} is known to both U and G and is the nym under which the permit is issued by G. G possesses a provable link Sig^{R} between the U's id and his Nym^{R}. This can be used in case of disputes.

The second pseudonym in the permit, Nym^{P}, is known to the user U only and is included in the permit as an attribute that is not known to G. This is done using a commitment, whereof U proves that he knows the corresponding *UserSecret* and Nym^{P} (underlined) such that $\text{Nym}^{\text{P}} \leftarrow \text{f}^{\text{G}}(\text{Nym}^{\text{R}}, \textit{UserSecret})$.

To obtain a new permit, after the previous one was lost, step 6 changes. After recalculating $\text{Nym}^{\text{P}} \leftarrow \text{f}^{\text{G}}(\text{Nym}^{\text{R}}, \textit{UserSecret})$ and generating a new commitment $\textit{Com}2 \leftarrow \mathsf{commit}(\text{Nym}^{\text{P}})$ (Step 4 and 5), U decrypts c, resulting in the opening info of the previous commitment. This allows U to prove that $\textit{Com}.\text{Nym}^{\text{P}} = \textit{Com}2.\text{Nym}^{\text{P}}$ (corresponds to step 6), convincing G that the same Nym^{P} was used.

Buying a Ticket. For each ticket order, U sends $\text{Nym}^{\text{E}} \leftarrow \text{f}^{\text{E}}(\text{Nym}^{\text{P}})$ to T and proves possession of the corresponding Nym^{P}. (4.7.c.1,2). The use of one-way function gives the user for each event a different, but event-unique nym. This gives T the possibility to limit the number of tickets per user while at the same time, this function avoids linking of T's customers to the customers of other Ts. Collusion with G does not help, because G does not even know Nym^{P}.

When ordering a ticket, the user proves that he is not blacklisted by showing *Restrictions*[EventType]. If U is blacklisted, he sends $\text{Nym}^{\text{T}} \leftarrow \text{f}^{\text{T}}(\text{Nym}^{\text{R}})$ to T and proves that Nym^{T} is correctly formed with $\text{Cred}^{\text{P}}.\text{Nym}^{\text{R}}$. T now looks up the exact restrictions associated with Nym^{T} on the blacklist (4.7.c.3). This limits linking possibilities and possible collusion with G. The latter is only useful for blacklisted Us.

The negotiation phase (4.7.c.4) requires the user's permit as input, such that RequestProof can be generated. RequestProof is a proof for G that U did request the negotiated tickets at the negotiated price. This proof is also deanonymizable by J which provably reveals Nym^{R}.

(4.7.a) Getting the first anonymous permit certificate Cred^{P}

(1)	U → G	:	authenticate(*eID*)
(2)	G ⇆ U	:	(Nym^{R}, Sig^{R}) ← generateSignedNym(*eID.NRN*)
(3)	G	:	Restriction[] ← getRestrictionBooleans(eID.*NRN*)
(4)	**U** ⇆ G	:	Nym^{P}← f^{G}(Nym^{R}, *UserSecret*)
(5)	U → G	:	(*Com*, OpenInfo) ← Comm(Nym^{P})
(6)	U → G	:	*Com*, prove(*Com*.Nym^{P} = f^{G}(Nym^{R}, *UserSecret*)), $c \leftarrow enc_{H(UserSecret)}$(OpenInfo)
(7)	U ⇆ **G**	:	Cred^{P}← issueCred(Nym^{R}, {*Com*.Nym^{P}, Restriction[], *attributes*})
(8)	G	:	store[*eID.NRN*, Nym^{R}, Sig^{R}, *Com*, *c*]

(4.7.a) Buying tickets

(1)	U → T	:	Nym^{E}← f^{E}(Cred^{P}.Nym^{P}), event
(2)	U → T	:	authenticate(Cred^{P}, {Cred^{P}.Nym^{P}≃Nym^{E}, Cred^{P}.Restriction[EventType]})
(3)	T	:	if(Cred^{P}.Restriction[EventType] = true) do
(3.a)	U → T	:	Nym^{T}← f^{T}(Cred^{P}.Nym^{R})
(3.b)	U → T	:	prove(Nym^{T}≃Cred^{P}.Nym^{R})
(3.c)	T	:	*Restrictions*← retrieveRestrictions($\text{Blacklist}_{\text{T}}$, Nym^{T})
(3.d)	T	:	end if
(4)	U ⇆ T	:	(SeatNb[], price, RequestProof) ← negotiate(Cred^{P}, event; Nym^{E}, eventPolicy, [*Restrictions*])
(5)	U ⇆ B ⇆ T	:	($\text{PayProof}_{\text{U}}$, $\text{PayProof}_{\text{T}}$) ← pay(price, Hash(SeatNb[], ...))
(6)	U ← T	:	tickets[] ← generateTickets(SeatNb[])
(7)	T	:	update [event, Nym^{E}, RequestProof, tickets[]]

Table 4.4: Protocols with anonymous credentials

Blacklist Maintenance and Retrieval. A law enforcement entity J forwards the court orders (*NRN*, *Restrictions*) to G. G substitutes the *NRN*s with the corresponding Nym^{R}s. Each $\text{Nym}^{\text{R}}$is further converted to Nym^{T}← f^{T}(Nym^{R}) before the blacklist is sent to a specific T to avoid linkabilities and profiling by T (4.5.b).

Misbehaviour and Deanonymization Protocol 4.5.c illustrates how the collaboration of E, T and G is required in order to obtain a (provable) link between the ticket and the user's id. The proof is (RequestProof, deanProof, Sig^{R}). If someone is put on a blacklist for EventType, his permit Cred^{P} is revoked. U can obtain a new Cred^{P}, with the updated restrictions booleans Restriction[EventType], immediately.

Evaluation

We now evaluate by checking the requirements

Functional and Security Evaluation

F1 Nym^{E}← f^{E}(Nym^{P}) enables T to link ticket orders of the same U for the same event.

(4.5.a) Maintaining the blacklists

(1)	J → G	:	Nym^{R}, Restrictions, EventType
(2)	G	:	Blacklists[EventType].add(Nym^{R}, *Restrictions*)
(3)	J → G	:	revokeCert(Nym^{R})

(4.5.b) Obtaining a blacklist

(1)	G	:	for each (Nym^{R}, *Restrictions*) in Blacklists[EventType]: $\text{Blacklist}_{\text{T}}$.add($\text{f}^{\text{T}}(\text{Nym}^{\text{R}})$, *Restrictions*)
(2)	T ← G	:	$\text{Blacklist}_{\text{T}}$

(4.5.c) Identifying buyer of a ticket

(1)	J ← E	:	complaint, seatNb
(2)	J → T	:	event, seatNb
(3)	J ← T	:	RequestProof← lookup(event, seatNb)
(4)	J	:	Nym^{R}, deanProof ← deanonymize(RequestProof)
(5)	J → G	:	(*NRN*, Sig^{R}) ← lookup(Nym^{R})

Table 4.5: Protocols with anonymous credentials (bis)

F2 A subscription can be issued by T or a coordinating organization. It can be an anonymous credential that contains Nym^{P}, Nym^{R}, the Restriction[EventType] booleans and information about the subscription. It can be anonymously shown to a ticketing service in order to obtain tickets without a payment phase. Alternatively, a multiple-use ticket with an expiry date can be issued.

F3 The user can selectively disclose properties in the permit.

F4 is explained in section 4.2.6.

F5 is done using the anonymized blacklists. Revocation of tickets issued to persons that were blacklisted after the ticket order is possible if Nym^{R} is systematically shown to T. However, the price is an increase in linkabilities.

Privacy Evaluation

P1 Deanonymization requires the collaboration of T, G and J as we argued in *Misbehaviour and Deanonymization*.

P2 We argued that a user has for each E a different $\text{Nym}^{\text{E}}\leftarrow \text{f}^{\text{E}}(\text{Nym}^{\text{P}})$. Nym^{P} is needed to do linking to other Es, but can only be obtained if both the user's secret *UserSecret* and the user's Nym^{R} are known. For blacklisted users, G can link Nym^{R} and Nym^{T}. Collusion of T and G is then possible.

P3 G knows the links between nyms on a blacklist and the user's id. However, such convictions are publicly available. Collusion of T and G can reveal the identity associated with Nym^{T}.

	Trivial	**Pseudonym certs.**	**Anon. creds.**
F1 - # Tickets	✓	✓	✓
F2 - Subscription	✓	✓	✓
F3 - Pricing	✓	✓	✓
F4 - Deanon.	✓	✓- J interacts with E, T, PS, G.	✓- J interacts with E, T, G.
F5 - Ban	—	✓+ ticket revocability	✓(2)
P1 - User anon.	T knows user id	If no collusion of E, T, PS, G. T knows permit atts.	✓
P2 - User profiles	T can link everything.	Linkability during limited, fixed period.	✓(1)
P3 - Anon. blacklists	—	If no collusion PS, G.	only G can identify. U.

(1): If the user is blacklisted, G can collude with one or more Ts.
(2): Ticket revocability is possible at the cost of increased linkabilites.

Table 4.6: Comparison of the three approaches

4.2.7 Evaluation

A comparison of the three approaches is given in table 4.6. It is clearly possible to fulfil the main functional/security requirements, while at the same time giving the privacy a serious boost. To maintain user-friendliness, the interactions with e.g. PS can be done transparently to the user. The anonymous credential based protocol is computationally the most intensive. Tests are needed to quantify this.

We have to be aware that the two proposed solutions disallow a banned person to buy tickets for someone else (e.g. father buys tickets for his children) and that it is still possible that a person buys tickets and gives them to a banned person. The solution is thus not perfect and still, police presence is needed on e.g. soccer matches.

4.3 ePoll

4.3.1 Introduction

In a poll, opinions of people are collected and processed. In paper-based polls the collection and processing takes a lot of time and effort. Electronic poll systems (ePoll), however, offer several benefits with respect to the paper-based polls. ePolls enable users to sign polls anywhere at any time and now reach wider sections of society. Moreover, automatic processing of the results can make the polls more reliable.

On the other hand, electronic poll systems introduce some new problems. Some systems are unreliable and may return incorrect results as for instance a user may sign a poll more than once. Other systems, use personal information to prevent multiple signing. However these systems are not privacy friendly.

This section presents PetAnon, a privacy-preserving poll system using Idemix anonymous credentials. PetAnon combines good privacy properties with reliable results.

Thi section is structured as follows. Section 4.3.2 gives the requirements of the system. Section 4.3.3 discusses the protocol used in PetAnon and is followed by section 4.1.4 with an evaluation of the protocol in respect of the requirements.

4.3.2 Requirements

The requirements of the privacy-preserving ePetition system are discussed below. They are classified according to security and privacy requirements.

Security requirements

S1 A user can sign a certain petition only once.

S2 A petition may possibly address only a subset of the potential signers; therefore the signer may be required to prove that he belongs to that subset.

S3 A user can verify that his signature is included in the petition's database.

S4 Everyone can verify the correctness of the petition results.

Privacy requirements

P1 Signers are anonymous.

P2 Signatures cannot be linked to a user. Moreover, signatures of different petitions cannot be linked to each other.

P3 A petition may request optional attributes that the user can release in order to get more differentiated results. The user has the choice if he wants to disclose these attributes or not.

4.3.3 Protocols

Roles and setting. A user U possesses an eID card, which is used when U authenticates towards the registration server R. This authentication is required before R issues a *voting credential* to U that can be used to sign an ePetition on a petition server P. The registration server R has a certificate containing the public key information used in the credential-issue and credential-show protocols.

Setting up an ePetition. The petition organizer P contacts R and offers to R the title, description and validity period of the petition. R generates a new, unique provable one-way function $f_{petition}(.,.)$ which needs two arguments. This function, as well as the user provided petition info are included in a (X.509) *petition certificate* $cert_{petition}$ that is issued by R to P. As a result, the latter obtains a corresponding private key $PK_{petition}$.

Retrieving a signPetitions credential. In order to sign a petition, U has to obtain a signPetitions credential. Therefore, he authenticates using his eID card (1). This actions reveals the personal data contained in the eID card to R.

Every citizen is only allowed to have one signPetitions credential. This is first checked by R. If the user did not register beforehand (2a), the user generates a (long) secure random number (2a.1), puts it in a commitment (2a.2), which is sent to R, and proves that he knows the committed value (2a.3). R also generates a (potentially shorter) random value (2a.4). These two random values will be used to prevent voting multiple times for the same petition, as we will see later.

If U previously had been issued a signPetitions credential (2b), these two random values are retrieved from R's storage and will be reused (2b.2). Also the credential's serial number is retrieved, which allows to revoke this credential before issuing a new one (2b.1).

After a serial number for the new credential is generated (3), all the parameters for the credential issuance are known and the signPetitions credential is issued (4). It contains the two random values, the serial number and a subset of the attributes (or properties thereof) that were extracted from the eID card. Note that R never gets hold of the user's secure random number.

Finally, U stores the credential (6), and R stores the commitment, the other random number and the serial number, as well as the user's NRN (National Registration Number) (5). This will allow R to check whether a user already has been issued a signPetitions credential, to revoke signPetitions credentials and to issue new ones.

Signing a petition. Initially, the petition server P authenticates using his petition specific certificate $cert_{petition}$. P additionally sends an overview of required and optional personal properties that must or can be proved when signing the credential. Each petition has certain required and optional attributes. For instance to sign a certain petition you must be older than 18 years. However, it is up to the user if he wants to reveal his gender or zip code. Finally, P sends a list of options for which the user can vote to U (1).

With the help of the petition's one-way function and the two random values contained in the vote credential, the user generates his petition specific nym (2), and sends it to P, together with the description of the personal properties that U is willing to disclose and the option for which he wants to vote (3).

Now, the interactive Idemix show protocol is run (4): U proves the selected properties, as well as that the petition specific nym for that user is correctly formed based on the random values contained in the credential. Thereby the user's vote choice is anonymously signed.

If that nym has not yet signed that specific petition (5), the protocol continues by generating a vote number. This is a reference to the petition-record that is being generated. The vote number, the hash of the proof and the user's nym are signed with the petition secret key, and stored by P together with that signature. The resulting record is made public. The signature is sent to and stored by U and allows U to check that his signature is included in the petition's database and to file a complaint otherwise and proof whether the record is changed by P.

Verification. The user can request from P the record with index voteNrand which was signed by the user. If the vote was tampered with, either the P-provided signature will no longer equal the signature stored by U, or the P-provided signature will no longer match the

$(proof, nym, voteNr)$-tuple made public by P.

If all the records are made publicly available, everyone can verify the correctness of the petition by verifying for each record the proof and the respective signature.

(4.7.a) Retrieving an anonymous credential

(1)	U → R	:	authenticate(eID)
(2a)	R	:	if (!credExists($eID.NRN$))
(2a.1)	U	:	secureRand← genSecureRand()
(2a.2)	U	:	(Comm, OpenInfo) ← commit(secureRand)
(2a.3)	U → R	:	Comm, prove({x | Comm== commit(x)}, Comm,OpenInfo)
(2a.4)	U → R	:	rand_R← genRand
(2b)	R	:	else
(2b.1)	R	:	(serialOld, Comm, rand_R) ← retrieveCredInfo($eID.NRN$)
(2b.2)	R	:	revokeCred(serialOld)
(3)	R	:	serial← genSerial()
(4)	U ⇆ **R**	:	Cred← issueCred(serial, Comm.secureRand, rand_R, subset(properties_{eID}))
(5)	R	:	store(serial, $eID.NRN$, Comm, rand_R)
(6)	U	:	store(Cred)

(4.7.b) Signing petitions

(1)	U ← P	:	authenticate($cert_{petition}$), $Options_{props}$, choices[]
(2)	U	:	$\text{Nym}_{\leftarrow}$ $cert_{petition}.f_{petition}$(Cred.secureRand, Cred.serial)
(3)	U → P	:	Nym, $props$ ← select($Options_{props}$), $choice$ ← $select(choices[])$
(4)	**U** ⇆ P	:	proof← showCred(Cred, $props$ && $\text{Nym}_0 \backsim$ Cred){choice}
(5)	P	:	if (petitionSigned(Nym) abort()
(6)	U ← P	:	voteNr← getVoteNr()
(7)	U ← P	:	receipt← Sig($SK_{petition}$, (voteNr, hash(proof), Nym_0))
(8)	P	:	store[voteNr, Nym_0, proof, receipt]
(9)	U	:	store[receipt, hash(proof), voteNr]

Table 4.7: Protocols for PetAnon

4.3.4 Evaluation

S1 is easily fulfilled, as for each petition the user is known by P under a petition specific nym. If that nym already signed that specific petition, the vote is cancelled.

S2 and [P3] are fulfilled. Some attributes in the credential show may be required by P, while others are up to U if he wants to disclose the information or not.

S3 is fulfilled. U can detect if his vote was tampered with based on the P-provided signature.

S4 If the records are made publicly, everyone can verify the correctness of the petition by verifying the proofs and signatures.

P1 Using the Idemix credential show protocol, as long as no identifying attributes are revealed, the user U remains anonymous, and different shows are unlinkable. Moreover, privacy is preserved in case of collusion of R and P.

P2 is fulfilled. To sign a petition, U authenticates anonymously using his credential (Cred). Signing the petition is done anonymously, and there are no identifiable actions linked to the signature.

4.4 Conclusion

The first application demonstrates how the eID card can be used for remote access over an untrusted network. The other applications merely use the Belgian eID card as a bootstrap to obtain privacy friendly tokens, enabling access to online services. The same strategy can be used for other applications in which privacy may be a concern such as physical access to buildings, age-verification, etc. The applications show the superior properties of anonymous credentials. However, their complexity and weak efficiency may currently impede applications with high performance requirements.

Chapter 5

eID extensions

The Belgian eID card has 3 main functionalities: revealing identity information, authentication and digital signature. However, these functionalities does not always comply with the requirements of the developer. Therefore, some eID extensions are presented. They use the eID card as a bootstrap mechanism to use more advanced credentials or to support more functionality (e.g. encryption, delegation of rights, etc.).

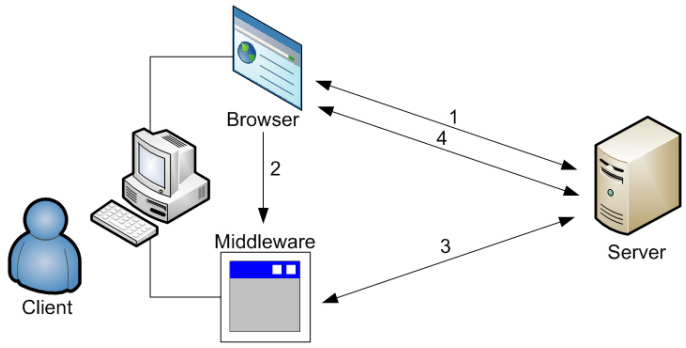
The first extension presents an alternative authentication method for the SSL client side authentication method. Proxy certificates are introduced in the second extension. The last extension provides a mechanism to store files (secrets, credentials, etc.) in an encrypted form on a file server using the eID card. Finally, these extensions are evaluated in a secure e-mail application.

5.1 Extension 1: eID authentication

Since single sign-on authentication can easily be abused by trojan horses, traditional client authentication (HTTPS) should be replaced by a new authentication protocol: auth.

Requirements

- The protocol should prevent trojan horses to authenticate secretly in the citizen's name and in addition require the user's consent for every authentication.
- The protocol should tackle man-in-the-middle attacks.
- The protocol should handle the authentication of sessions between a client and a web server. This makes it easy to integrate with web access.
- The protocol must be usable with every authentication mechanism that is based on a challenge and signature scheme. It should be compatible with all kinds of eID cards that allow for authentication based on a challenge-response protocol.
- The protocol only requires the presence of the eID card for a short time. After successful authentication, the card can be removed from the card reader.



1. There exists an HTTP session between client and server.
2. User clicks on ***auth*** URL.
3. User authenticates to server and gives a reference of the HTTP session.
4. User can access private content in existing HTTP session.

Figure 5.1: Connection flow of ***auth*** protocol

Description and implementation

Figure 5.1 shows the message flow of the ***auth*** protocol. First, the user browses to a web page containing an ***auth***-URL. The URL has the following syntax:

```
auth://SPhost/path/auth-web-service#sessionID&types
```

A new HTTP session is started by the web server when this web page is requested. Each session has an unique number (*sessionID*). This number can be either stored in a cookie on the user side or can be propagated in the URL. When clicking on the ***auth***-URL, the module in the middleware that handles the ***auth*** protocol is executed. The middleware will pop-up a warning window that shows the (domain) name of the other party, it will invite the user to select an authentication means from the list of supported *types* (e.g. an eID card) and to give his consent for the authentication. Sometimes, the user will have to activate the authentication means by entering a PIN code or a password. The middleware module will set up a separate HTTPS connection with the web server and ask for a challenge. Validation of the server certificate is important when setting up this connection to prevent man-in-the-middle attacks. Next, extra information is appended to the challenge (the type of authentication means, the domain name of the SP, the *sessionID* and a *middleware_secret*) and the aggregate is signed with the authentication key. The *middleware_secret* is a secret message which is programmed in the middleware code and hidden through code obfuscation. The generated signature is sent together with the certificate chain of the used "authentication" keypair to the web server. The middleware module on the server verifies the signature and the certificate chain. Then the server needs to check the equality of the IP addresses used by the client in the browse-session and in the authentication session. If all the checks pass, the browse-session is converted into an authenticated session and the user can browse to private pages on the website. The session is active until the user logs out or the session times out. Table 5.1 contains a detailed description of the ***auth*** protocol. The pop-up message described in (5.1.b.12) can be avoided by including a JavaScript program in the HTML-page with the ***auth***-URL, which continuously polls for the termination of the ***auth*** protocol.

(5.1.a) Browse to login page

(1) U → SP : HTTPS request
: Header: [GET /login.html — Host: *SPhost*]
(2) SP : startSession(*sessionID*; IP_{client}; *challenge*; $< now + timeout >$)
(3) U ← SP : HTTPS response
: Header: [Set-Cookie: session=*sessionID*; Expired=$< now + timeout >$; secure]
: Body: [HTML page with hyperlink to "auth://*SPhost*/auth.php#*sessionID*&*types*"]

(5.1.b) Authentication over a new connection

(1) U : Request user's consent with pop-up window
: [Do you want to authenticate with this authentication *type* to *SPhost*?]
(2) U : if(*user_input* == "yes"){select authentication means and activate it} else {abort}
(3) U → SP : HTTPS request
: Header: [POST /*path*/auth.php?session=*sessionID*&type=*typeID* — Host: *SPhost*]
: Body: ["getChallenge"]
(4) U ← SP : HTTPS response
: Body: [*challenge*]
(5) U : $signature = \text{sign}_{SK_{auth}}(< type \parallel challenge \parallel SPhost \parallel sessionID \parallel$
: $middleware\_secret >)$
(6) U → SP : HTTPS request
: Header: [POST /*path*/auth.php?session=*sessionID* — Host: *SPhost*]
: Body: [response=*signature*&authCertificate=$CertChain_{auth}$]
(7) SP : if(IP_{client} != lookupIP(*sessionID*)) abort
(8) SP : if(validateCertificate($CertChain_{auth}$) == false) abort
(9) SP : if(verify$_{PK_{auth}}$(*signature*; $< type \parallel challenge \parallel SPhost \parallel sessionID \parallel$
: $middleware\_secret >$) == false) abort
(10) SP : setSessionAuthenticated(*sessionID*; *signature*; $< now + timeout >$)
(11) U ← SP : HTTPS response
: Body: ["OK"]
(12) U : Show pop-up to user that authentication is performed successfully.

(5.1.c) Browse to private content

(1) U → SP : HTTPS request
: Header: [GET /private/index.html — Host: *SPhost* — Cookie: session=*sessionID*]
(2) SP : if(isAuthenticated(*sessionID*) == false) abort
(3) U ← SP : HTTPS response
: Header: [Set-Cookie: session= *sessionID*; Expired=$< now + timeout >$; secure]
: Body: [< content of requested page >]

Table 5.1: The ***auth*** protocol

Modifying the card

The ***auth*** protocol can be implemented by only adapting the current middleware. However, the security can be increased by having the card implement the ***auth*** protocol. Currently, the eID card receives a challenge from the middleware and signs it after the PIN is entered. If other parameters like *sessionID* and *SPhost* can also be sent to the card for authentication, the card can compose the message that has to be signed (like in 5.1.b.5). The *middleware_secret*

can then be omitted, since the card will only sign when the user has given his consent (OK button or PIN code).

5.2 Extension 2: Proxy certificates

5.2.1 BeID proxy certificates

Although the eID card itself cannot encrypt nor decrypt data, the right to do this can be delegated to the system to which the card reader is connected. This restricted delegation and proxying to another entity can be achieved through proxy certificates [58].

In the sequel we will abbreviate the phrase *"signed with the private key that corresponds to the public key certified in a certificate"* into *"signed with the certificate"*.

Proxy certificates. A proxy certificate is an extended version of a normal X.509 certificate. Proxy certificates are derived from and, hence, signed by a normal X.509 certificate or by another proxy certificate. For every proxy certificate, a new key pair is generated; the new public key is included in the proxy certificate. The proxy certificate and its corresponding private key can be used for asymmetric encryption resp. decryption.

Modified standard. The standards for proxy certificates, as defined in the RFC 3820 [58], present some problems when used in the context of the Belgian eID. Therefore, some modifications have to be made (see figure 5.2). First, according to the RFC, proxy certificates should be signed with the authentication certificate, since only this certificate of the Belgian eID contains the required value for the *key-usage* attribute[1]. The *key-usage* attribute defines the purpose of the corresponding private key. However, using the authentication key SK_{Auth} is less secure than using the signature key SK_{Sig} since the PIN is only required for the first authentication, while it is required for every signature. Therefore, we propose to use SK_{Sig} for signing proxy certificates. Second, Belgian legislation prohibits to store the *NRN*. However, the subject field contains both the owner's name and his *NRN*. This implies that the eID certificates may not be stored. Nevertheless, the proxy certificate standard specifies that the subject of the issuing certificate is to be copied into the issuer and subject fields of new proxy certificates. To solve this problem, we propose to copy the name of the owner into the subject field and to copy the hash of the *NRN* into the issuer field instead of the subject of the eID certificate. For validation purposes, the serial number of the issuer certificate is also included in the certificate (i.e. *issuerSN*). Additionally, another extra attribute indicates the certificate type: `BEID-PROXY`. In this scheme, we assume that when the eID certificate is revoked (e.g. in case of loss or theft), all issued proxy certificates are also no longer valid. Moreover, every proxy certificate must expire before the expiration date of the issuing eID certificate.

The protocol in table 5.2 demonstrates the creation of a BeID proxy certificate. The user U generates an asymmetric key-pair (1). A serial number is generated from the hash of the *NRN* and the *issueDate* of the new proxy certificate (2-3). The *NRN* in the hash avoids collisions, while the *issueDate* enables users to have more than one proxy certificate. A proxy certificate *proxyCert* is then generated and signed with the eID card (4-5). Finally, the signature is inserted in the certificate.

[1] In the Belgian eID card, the authentication certificate has as key usage *Digital Signature*, while the signature certificate contains *Non-repudiation*!

	Belgian signature Certificate	*Proxy Certificate*
SerialNumber:	*5874.....2345*	*Hash($cert_{sig}$.SubjectName.NRN || IssueDate)*
SubjectName:	*FullName; NRN; ...*	*FullName*
SubjectPK:	*52:05:11:21:...:d2:7b*	*23:2b:24:a4:... :93:c5*
ExpiryDate:	*expiry date*	*$cert_{sig}$.ExpiryDate*
IssueDate:	*xx:xx:xx xx:xx*	*xx:xx:xx xx:xx*
CRL:	*crl.eid.belgium.be/...*	*crlLocation*
Issuer:	*Citizen CA; BE; ...*	*Hash($cert_{sig}$.SubjectName.NRN)*
Signature:	*15:f5:55:ff:...:20:f6*	*d5:fe:23:b4: ... :44:ab*
Extensions:		
IssuerSN:	*[not present]*	*$cert_{sig}$.SerialNb*
Type:	*[not present]*	*BEID-PROXY*

Figure 5.2: Content of the modified proxy certificate.

createBeIDProxy([***attributes***]):

(1)	U	:	$(SK_U, PK_U) \leftarrow$ generateKeyPair()
(2)	U	:	$issueDate \leftarrow$ getDate()
(3)	U	:	$serialNb \leftarrow$ hash(NRN || $issueDate$)
(4)	U	:	$proxyCert \leftarrow$ generateProxy(BEID-PROXY, $cert_{Sig}$, $serialNb$, PK_U, $issueDate$, $crlLocation$, [$attributes$])
(5)	$U \leftrightarrow C$	:	$Sig \leftarrow$ $\text{sign}_{\text{eID}}(\text{hash}_{\text{M}}(proxyCert))$
(6)	U	:	$proxyCert.Sig \leftarrow Sig$

Table 5.2: Create a new proxy certificate.

Revocation of proxy certificates. A special purpose server R_P can keep track of revoked proxy certificates and publish the appropriate CRLs. To revoke a proxy certificate (cfr. table 5.3), the user authenticates with his eID card (1) and sends the proxy certificate he wants to revoke (2). If the issuer corresponds to the eID signing certificate (i.e. hash($cert_{Sig}.NRN$) = $proxyCert.Issuer$), the proxy certificate is revoked by adding its serial number to the latest CRL.

revokeBeIDProxy(***proxyCert***):

(1)	U $\leftrightarrow$ R_P	:	$NRN \leftarrow$ $\text{authenticate}_{\text{eID}}()$
(2)	U $\rightarrow$ R_P	:	revokeCertificate($proxyCert.serialNb$)
(3)	S	:	if (hash($cert_{Sig}$.SubjectName.NRN) $\neq$ $proxyCert$.Issuer) abort
(4)	U $\leftarrow$ S	:	true $\leftarrow$ addToCRL($proxyCert.serialNb$)

Table 5.3: Revoke a proxy certificate account.

Validation. A receiver validates a new proxy certificate by checking its signature, the validity period, its revocation status and by verifying the rest of the certificate chain. Since the hash of *NRN* is kept in the issuer field, name chaining (cfr. RFC 3280 [47]) for certification path validation will fail. However, the extra attribute *issuerSN* included in the certificate

binds the eID certificate to the proxy certificate. The first step in creating the certification path needs thus to be modified: the serial number of its issuing eID certificate must match the *issuerSN* in the proxy certificate.
To comply with Belgian legislation, the eID certificate is deleted after validation. Hence, future validation is not possible. However, verifying the validity period and the revocation status suffices as the proxy certificate was stored on a trusted location after its validation. Additionally, the revocation status of the eID certificate can be verified by checking the *issuerSN* of the proxy certificate in the CRLs of the Belgian eID.

The scheme described in this section allows for creating legitimate *proxy certificates* (created with the eID card), that can be used in many applications. Moreover, once a proxy certificate has been created, the Belgian eID is no longer required.

5.2.2 Evaluation

The *proxy certificate* mechanism allows owners of an eID card to setup mutually authenticated secure channels without the need for a trusted third party. Secure communication is even no longer restricted to SSL. The proposed system with proxy certificates makes it more flexible and extensible. Although not completely complying with the standards, the proposed scheme supports asymmetric *encryption* with the eID card. Moreover, the proxy certificates can be used for asynchronous communication (i.e. recipients can decrypt confidential messages after the communication channel has been closed).

Once the receiver of a proxy certificate has deleted the eID certificate that signed the proxy certificate (as is imposed by Belgian legislation), he can still check the validity of the proxy certificate and the revocation status of the eID certificate. Although other certificates in the validation path (i.e. *Citizen_CA*, *Belgium_Root_CA*, ...) may have been revoked, the proxy certificate can include the serial numbers and the CRL-locations of these certificates as extra attributes to make the verification of the rest of the certificate chain possible. Note that optional attributes in the proxy certificate may further restrict its use.

The storage of the private key corresponding to the public key in the proxy certificate requires more trust in the device storing it. A developer must pay special attention on how to use this technology. For instance, when using proxy certificates on mobile phones, the developer should store the private key inside the SIM card. Moreover, revocation and a limited validity period may reduce the implications of a stolen private key. Anyhow, revoking a proxy certificate causes less burden than revoking eID certificates.

5.3 Extension 3: Secure storage of secrets

A major challenge in today's society is to enable access to confidential data (e.g. personal information, secret keys, ...) from various locations. Data must be securely stored on an easily accessible remote server. Data is typically encrypted by the owner before it is sent to and stored on the server. Hence, the encrypted data is useless to the server or any adversary that may have access to it. In this section a scheme is presented –based on the Belgian eID card– for securely storing confidential data.

5.3.1 Construction

A trivial solution consists of encrypting the data with the public key of the authentication certificate. Whenever needed, the cipher text is fetched from the server and decrypted using the corresponding private key. However, encryption/decryption with the Belgian eID card is not possible. Moreover, when the card is lost or renewed, decryption of previously encrypted information is no longer possible. Therefore, a new mechanism is proposed that uses the Belgian eID card as a bootstrap.
A secret (symmetric) key is derived from a signature generated by the eID card. This secret key is used for encrypting data before it is stored. When a user wants to retrieve his confidential information, the encrypted information is fetched from the server and decrypted with the secret key, which is regenerated using the same eID card. In the following paragraphs, the protocols for storing and retrieving confidential data are discussed in more detail.

storeConfidentialData(***data***, ***tag***):

(1)	U	:	$keyGenMsg \leftarrow$ '*KEYGEN*' $\|\|$ NRN $\|\|$ $CardNb$ $\|\|$ $otherUserInfo$
(2)	$U \leftrightarrow C$	:	$Sig \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}_{\mathsf{M}}(keyGenMsg))$
(3)	U	:	$K_T \leftarrow \mathsf{createSymKey}(PRG, \mathsf{hash}_{\mathsf{T}}(Sig))$
(4)	U	:	$E_{tag} \leftarrow \mathsf{encrypt}(tag, K_T)$
(5)	U	:	$R \leftarrow \mathsf{generateRandom}()$
(6)	U	:	$K_D \leftarrow \mathsf{createSymKey}(PRG, \mathsf{hash}_{\mathsf{D}}(Sig \,\|\| R))$
(7)	U	:	$E_{data} \leftarrow \mathsf{encrypt}(data, K_D)$
(8)	$U \leftrightarrow S$	:	$NRN \leftarrow \mathsf{authenticate}_{\mathsf{eID}}()$
(9)	$U \rightarrow S$	:	(E_{data}, R, E_{tag})
(10)	S	:	$index \leftarrow \mathsf{hash}_{\mathsf{L}}(NRN \,\|\| E_{tag})$
(11)	S	:	$\mathsf{store}([E_{data}, \mathrm{R}]; index)$
(12)	S	:	delete E_{tag}
(13)	$U \leftarrow S$	:	$(Timestamp, \mathsf{sign}_{\mathsf{S}}(\mathsf{hash}_{\mathsf{R}}([Timestamp, E_{tag}, E_{data}, R])))$

Table 5.4: Storing confidential data remotely.

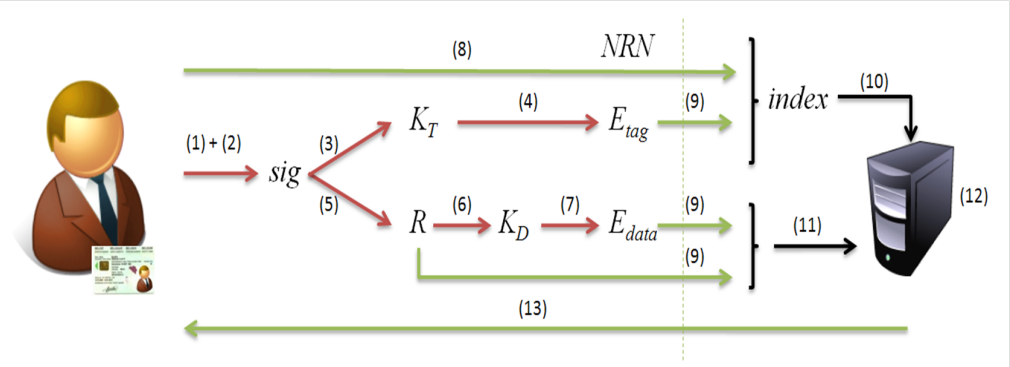


Figure 5.3: Storing confidential data.

Storing confidential data. Table 5.4 and figure 5.3 show the steps for storing confidential data. The user provides the *data* to be stored and a secret *tag* that serves as a name or alias

of the data.
First, the Belgian eID card is used to generate a secret key as follows. A message *keyGenMsg* with a fixed format is constructed (1). The message consists of a header '*KEYGEN*', the *NRN*, the serial number of the eID card and possibly some other user information, making the message card specific. The hash of the message, $\mathsf{hash}_\mathsf{M}(keyGenMsg)$ is then signed with the signature key on the eID card resulting in a 1024 bits signature *Sig* (2). The fixed format prevents that an adversary obtains *Sig* by requesting a signature on a forged message; any message to be signed by the eID card should not match this format, except in this protocol. Subsequently, *Sig* is used as the seed for generating two new symmetric keys, namely K_T and K_D. To ensure that the same keys are generated, independent of the platform, a specific pseudo-random generator *PRG* is passed as a parameter of the createSymKey function; also, the hash-functions used should be fixed. K_T is used for encrypting the secret *tag* associated with the data. The key is derived from the hash of *Sig* (3-4). The encrypted tag E_{tag} is used in step 10 of the protocol to derive an *index* to a record in the server database. K_D is used for encrypting the *data*. Each time information is stored (i.e. each time the protocol is invoked), a new random number R is associated with it (5). From the hash of R and *Sig* the encryption key K_D is derived (6) with which the *data* is encrypted into the cipher text E_{data} (7). In each execution of storeConfidentialData the data will be encrypted with another key; this prevents linking of the same encrypted data that is stored under multiple tags.
Next, the encrypted data E_{data} is stored on a remote server S. First, the user authenticates with his eID, to ensure that later only the owner U can retrieve his uploaded data (8). Then, U sends the encrypted tag E_{tag} and the encrypted data E_{data} to S (9). The *NRN*, disclosed during the authentication, is hashed together with E_{tag} and will be used as an *index* to the information that is stored in the server database (10). The use of the encrypted secret tag, E_{tag}, ensures the user's privacy (*index obfuscation*), while making the *index* tag specific. A different *tag** will result in a different index. Finally, the database stores a record with the encrypted data and the random number R at location *index* (11). Although useless to the server or any other adversary, the random number R is necessary for the owner of the data to regenerate the correct symmetric key for decrypting E_{data}. E_{tag} is deleted permanently (12). If the server S is trustworthy (i.e. it permanently deletes the E_{tag} immediately after storing E_{data}), dictionary attacks on the *index* are no longer feasible. An adversary with full access to the server data cannot link any data to a particular citizen.
A receipt is sent to the user, certifying the proper storage of the encrypted data (13).

retrieveConfidentialData(***tag***):			
(1)	U	:	$keyGenMsg \leftarrow$ '*KEYGEN*' $\|\|$ *NRN* $\|\|$ *CardNb* $\|\|$ *otherUserInfo*
(2)	$U \leftrightarrow C$	:	$Sig \leftarrow \mathsf{sign}_\mathsf{eID}(\mathsf{hash}_\mathsf{M}(keyGenMsg))$
(3)	U	:	$K_T \leftarrow \mathsf{createSymKey}(PRG, \mathsf{hash}_\mathsf{T}(Sig))$
(4)	U	:	$E_{tag} \leftarrow \mathsf{encrypt}(tag, K_T)$
(5)	$U \leftrightarrow S$	:	$NRN \leftarrow \mathsf{authenticate}_\mathsf{eID}()$
(6)	$U \rightarrow S$	:	$\mathsf{requestRecord}(E_{tag})$
(7)	$U \leftarrow S$	:	$record \leftarrow \mathsf{getRecord}(\ \mathsf{hash}_\mathsf{L}(NRN \ \|\| \ E_{tag}))$
(8)	S	:	delete E_{tag}
(9)	U	:	$K_D \leftarrow \mathsf{createSymKey}(PRG, \mathsf{hash}_\mathsf{D}(Sig \ \|\| \ record.R))$
(10)	U	:	$data \leftarrow \mathsf{decrypt}(record.E_{data}, K_D)$

Table 5.5: Retrieving confidential data remotely.

Retrieving confidential data. When the data has been stored on the remote server, it can be retrieved by the owner from everywhere (see table 5.5). First, the message *keyGenMsg* is reconstructed (1) and signed with the eID card (2). With the signature *Sig*, the secret key K_T (3) is regenerated for encrypting the secret tag (4). Next, the user authenticates to the server S with his eID card (5). The encrypted secret tag, E_{tag} is sent and the corresponding record is requested (6). The server S fetches the record with $index = \mathsf{hash_L}(NRN \mid\mid E_{tag})$ from his database; the record, comprising of encrypted data and the random number, is sent to U (7). To ensure privacy, E_{tag} is permanently deleted (8). U can now regenerate the data specific secret key K_D from the hash of *Sig* and R (9). Finally, U decrypts E_{data} with the secret key K_D (10).

Recovery of lost keys. This scheme exploits the property that a deterministic signature algorithm is implemented in the eID card. When creating a signature with the eID card, the same input (*keyGenMsg*), always results in the same signature *Sig*. As such, using the same *PRG* and the same hash-functions ($\mathsf{hash_T}$, $\mathsf{hash_D}$, $\mathsf{hash_L}$), the same secret keys K_T and K_D (for a certain R) are always regenerated. However, in case of loss or renewal of the eID card, PK_{Sig}, SK_{Sig} and *CardNb* will have changed and the generated signature *Sig** no longer matches the signature *Sig* (generated by the previous eID card), impeding the localization and decryption of the stored information. Therefore, to protect important data, a secure backup of the signature *Sig* is created the first time this signature is generated. In case of loss or renewal, *Sig* is restored from the backup and the keys can be recovered. This secure backup could be provided by a key escrow service. The secret *Sig* is then split into n parts using a secret sharing algorithm and each part is stored on a different escrow server. To reconstruct the secret, all n parts are retrieved from the escrow servers and the interpolation of the parts results in the secret. To ensure that users only obtain their own keys, eID authentication can be used with the escrow servers [56, 21]. Additionally, a hash of the *NRN* and the serial number of the eID card can be used as an index to store a part of the secret. Every citizen can –online– lookup his current and previous card numbers at the National Registry. With this scheme, the capabilities of the eID card (authentication and signing) are extended with a mechanism to store and retrieve confidential data without the need to have a secure key management mechanism. The user only needs his card to access the encrypted data that is stored on the remote server. However, adversaries might try to get hold of the secret keys by continuously sending challenges to the eID card. Our solution tackles the threat by using the signature key in the eID card. Signing requires a PIN for every signature while authentication requires only a PIN the first time (Single Sign-On feature of the eID card). Moreover, the fixed format of the message allows to detect trojan horses or malicious applications that request users to sign a message resembling the proposed format.

To support recovery of confidential data when the eID card is lost or has become invalid, a secure backup of the signature *Sig* needs to be created the first time this signature is generated. However, the user remains responsible for making the secure backup. An alternative approach is to support a key-escrow mechanism.

5.4 Applications

The extensions discussed above promise new opportunities for the Belgian eID technology. The proposed extensions can be used in a variety of applications. The following applications demonstrate their reuseability. However, depending on the application, some fine tuning may be required.

5.4.1 Secure Storage Applications

A Local Secure Vault

Workstations are often used by multiple users. In a home environment, a laptop is used by different members of the family. In a work environment, a computer may be shared by multiple colleagues. It is often not very difficult to access other users' confidential data (such as passwords, secret keys, ...) unless appropriate countermeasures are taken.
This application uses the secure storage scheme for the implementation of a local credential vault. As such, users do not have to remember their passwords anymore (except for the PIN code of their eID card), which allows them to select more secure passwords. Every user has the exclusive right to access his own credential vault to store, access and update confidential data. In a home environment, index obfuscation may be omitted, hence, the scheme can be simplified.

Shared Confidential Files

With some small modifications to the secure storage extension, it is possible to have shared access to securely stored information. Hence, it is easy to build closed user groups in a corporate environment (e.g. managers can share top secret documents, development teams and focus groups can share confidential project information, ...). The data are ubiquitously accessible and no secrets are stored.

We assume a common file server S (with access control mechanisms). For every *shared encrypted file*, "F", a companion file, "*.F.keys*", is generated which contains for every user U_i with access to F a tuple $[U_i, R_i, \Delta_i]$ which is used to recreate the symmetric key with which F has been encrypted. The companion file will be maintained (i.e. can only be modified) by the owner of the shared encrypted file. However, the companion file does not contain secrets and, hence, does not need to be kept confidential except maybe for privacy reasons (to hide who has access to a file).

Decryption. The scheme works as follows (cfr. Table 5.6): U_i retrieves his tuple from the companion file (1) and generates a signature on a fixed message with his eID card (2-3). Then, the signature, the file's name "F" and R_i are hashed into a fixed-length bit string which is xor-ed with Δ_i to serve as a seed for the key generation function (4). The generated key K can be used to decrypt the file F or to encrypt the modified content.

keyGenSharedFile("F"):			
(1)	$U_i \leftarrow S$	:	$(U_i, R_i, \Delta_i) \leftarrow \mathsf{readTuple}(".F.keys", U_i)$
(2)	U_i	:	$keyGenMsg_i \leftarrow$ '*SHARED-KEYGEN*' $\|\|$ NRN_i $\|\|$ $CardNb_i$ $\|\|$ *otherUserInfo*
(3)	U_i	:	$Sig_i \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}_{\mathsf{M}}(keyGenMsg_i))$
(4)	U_i	:	$K \leftarrow \mathsf{createSymKey}(PRG, \mathsf{hash}_{\mathsf{S}}(Sig_i \,\|\|\, "F" \,\|\|\, R_i) \oplus \Delta_i)$

Table 5.6: Decrypting a shared encrypted file

Setup. The owner of the shared file (U_o) generates a randomly chosen *seed*, which will be used for generating a secret symmetric key K with which the file F can be encrypted and decrypted. The scheme works as follows (cfr. Table 5.7):

U_o first selects a random value R_o and a random *seed* (1-2), generates a signature on a fixed message (which is eID card specific) (3-4). Then, the bitwise difference between the *seed* and the hash of the signature, the file's name "F" and the random value R_o is computed (5). Finally, the companion file is created with content the tuple [U_o, R_o, Δ_o] (6).

sharedKeySetup("F"):			
(1)	U_o	:	$R_o \leftarrow \mathsf{generateRandom}()$;
(2)	U_o	:	$seed \leftarrow \mathsf{generateRandomSeed}()$;
(3)	U_o	:	$keyGenMsg_o \leftarrow$ '*SHARED-KEYGEN*' $\|\|$ NRN_o $\|\|$ $CardNb_o$ $\|\|$ *otherUserInfo*
(4)	U_o	:	$Sig_o \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}_{\mathsf{M}}(keyGenMsg_o))$
(5)	U_o	:	$\Delta_o \leftarrow seed \oplus \mathsf{hash}_{\mathsf{S}}(Sig_o \,\|\|\, "F" \,\|\|\, R_o)$
(6)	$U_o \rightarrow S$	:	$\mathsf{createKeyFile}(".F.keys", [U_o, R_o, \Delta_o])$

Table 5.7: Setting up the companion file with key material

Extending the set of sharing users. A new user U_j ($j > i$) can be added to the companion file by U_o via a similar scheme (cfr. Table 5.8). The owner U_o invites user U_j to send the necessary information to update the companion key-file (1). U_j will generate a random value R_j and sign a fixed message with his eID card (2-4). The signature is hashed with the file name "F" and the random value R_j (5). Then, R_j and H_j are sent to the owner over a secure channel (6) who will xor H_j with the secret *seed* (8) and update the companion file (9).

sharedKeyExtension("F"):			
(1)	$U_j \leftarrow U_o$	:	prepare for sharing "F"
(2)	U_j	:	$R_j \leftarrow \mathsf{generateRandom}()$;
(3)	U_j	:	$keyGenMsg_j \leftarrow$ '*SHARED-KEYGEN*' $\|\|NRN_j$ $\|\|$ $CardNb_j$ $\|\|$ *otherUserInfo*
(4)	U_j	:	$Sig_j \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}_{\mathsf{M}}(keyGenMsg_j))$
(5)	U_j	:	$H_j \leftarrow \mathsf{hash}_{\mathsf{S}}(Sig_j \,\|\|\, "F" \,\|\|\, R_j)$
(6)	$U_j \overline{\rightarrow} U_o$	:	R_j, H_j
(7)	U_o	:	$\Delta_j \leftarrow seed \oplus H_j$
(8)	$U_o \rightarrow S$	:	$\mathsf{appendKeyFile}(".F.keys", [U_j, R_j, \Delta_j])$

Table 5.8: Extending the set of users

Note that in this scheme, the only secret that needs to be sent over the network is H_j: if an eavesdropper gets hold of that value, and later fetches Δ_j from the companion file, he could easily construct *seed* and, hence, generate the secret key. A solution using proxy-certificates is presented in section 5.4.3. Of course, any other means to securely transfer that value from U_j to U_o could also be used (e.g. transfer via memory stick).

The user of random values R_i is necessary to allow for the removal of users from the set or to allow for renewing the encryption key of the confidential file. Otherwise, the users who have been removed from the set, could easily compute the file's encryption key.

Evaluation Sig_i is secret and never leaves the user's workstation. If an adversary does get hold of the Sig_i of a group member, the user must change *otherUserInfo*. For instance, it may include an updated version number. However, all files shared by the user and all corresponding key files must be updated (new keys must be chosen). Additionally, a GroupID can be included into *otherUserInfo*. This ensures a different *Sig* per group. However, the PIN-code will have to be entered multiple times when files belonging to different groups are consulted. The hash $\mathsf{hash}_5(Sig_i \,||\, "F" \,||\, R_i)$ is file and version specific. Hence, all group members can compute it, but it can never be used to access another file. This prevents a user with knowledge of the seed and encryption key of one file, to calculate the encryption key of another file.
The scheme is efficient and user-friendly. As Sig_i is group specific, the PIN is required only once for every group involved. The same Sig_i is used to generate keys for every file that is shared by the same group.
Note that the access control mechanism of the file system can limit the consequences of a breach of Sig_i.

5.4.2 Proxy

The Belgian proxy certificates scheme bootstraps lots of practical applications. The use of an existing nation-wide Belgian PKI infrastructure, maintained by the government considerably decreases the implementation costs of applications requiring strong security. Moreover, rights can be delegated to other devices and/or individuals. Constraints (e.g. purpose, location, time intervals, ...) and liabilities can be included in proxy certificates.

Proxy certificates may ease interoperability of eID infrastructures of different countries although legislation may impose restrictions. The Belgian eID certificate needs to be stored to verify the validity of a proxy certificate which is not allowed by Belgian legislation. As a result, the scheme cannot be used for public services. For instance, creating proxy a certificate that serves as a server certificate for a secure public website is not possible since its validity cannot be verified. Moreover, it is important to carefully handle the private keys associated with the proxy certificates. Below, three different application domains are presented in which the proxy certificates can be used, namely delegation of rights to another device, delegation of rights to another person and secure communication channels.

Delegation to another device

The GSM trade association has announced that the number of mobile devices has passed the 4 billion mark in December 2008. This number represents more than 60 percent of the world's total population. The number of portable Internet-enabled devices are set to grow to 1.5 billion by 2012. Experts anticipate that by 2020, mobile phones will be the primary Internet

devices for most people in the world. It is to be expected that mobile devices will become the main guardians and managers of our multiple electronic identities for a broad range of applications and services which include payments, e-health, e-government, etc. The Belgian eID card may be used to delegate electronic identities to mobile devices. Hence, Belgian proxy certificates can be stored and used on mobile devices to secure access to corporate email and resources, personal health data or e-government applications based on a nation wide infrastructure.
A major reason to implement single-sign-on for authentication is that a private key operation on the card is required to renew the session key during an SSL session. The server typically fixes the time interval between two session renewals. However, single-sign-on results in severe security and privacy threats using the Belgian eID card as discussed in [60]. Proxy certificates may solve this problem as the user may temporarily delegate the right to access that specific service to the browser.
Further, it supports physical access. A building automation system may allow access to a building or secure area based on the identity of individuals. Using proxy certificates, their identity can be proven easily.

Delegation to other individuals

A second application domain is where one person delegates his rights to another individual by means of a proxy certificate to allow that individual to sign or authenticate in the name of the delegator. The following are two examples:
A first example concerns online-tax-declaration. Currently, online-VAT-declaration requires a digital signature based on the Belgian eID of the employee submitting the declaration, instead of the director of the company. However, using a proxy certificate, the employee can declare the taxes *in the name* of the director. Moreover, the proxy certificate can include restrictions. For instance, it may include that it is only valid for vat-declarations, during office hours, by a specific employee.
A second example in which this kind of delegation may be useful is for delegating privileges such as physical access to another person.

Secure communication

In contrast with the Belgian eID card, proxy certificates facilitate encryption and decryption. As such, based on an existing nation-wide PKI infrastructure, it is possible to implement peer to peer SSL communication. For instance, in a secure chat application, trust can be established between two persons based on these proxy certificates, as both are signed with the eID card.

5.4.3 Combined

A combination of both schemes is shown below.

Secure Email

Combining both extensions enables the development of a secure email application based on the Belgian eID technology. It supports exchanging confidential e-mail messages using the BeID proxy mechanism described in section 5.2. Moreover, individuals can use the e-mail

service at any location as the necessary key material and contact information are stored securely on an easy accessible remote server.

Setup Every user generates a BeID proxy certificate, sends it to his contacts and retrieves a proxy certificate from every contact. This procedure contains the validation of the eID certificate as well as the proxy certificate itself. Next, The user contacts his remote secure storage and stores the proxy certificates of his contacts, together with his own proxy certificate and corresponding private key.

Sending and receiving messages When a user wants to send an email message, he generates a symmetric key SK and encrypts the message M. Next, for every recipient, he fetches the corresponding proxy certificate, which he retrieved earlier, from his remote secure storage. The key SK is encrypted with the public key of each of these proxy certificates. As a result, there are as much encrypted secret keys as there are recipients (i.e. $E_{SK-contact1}, E_{SK-contact2}, \ldots$). Finally, the encrypted message, together with the encrypted keys is sent to the recipients.
When a user receives an encrypted message, he selects his personal proxy certificate used to encrypt the symmetric key and fetches the corresponding private key from the secure store. Finally, the symmetric key is decrypted and used for decrypting the message.

Extended Shared Secure Storage

To enhance the security of the shared secure storage scheme discussed above, proxy certificates are introduced. The secret H_j may be sent over the network. An adversary may use this information to break the security of the scheme. Therefore, before sending the hash H_j, it is encrypted with the public key of the owner's (U_o) proxy certificate. The modified sharedKeyExtensionprotocol from table 5.8 becomes:

sharedKeyExtension("F"):

(1) $U_j \leftarrow U_o$	:	prepare for sharing "F", $proxyCert_o$
(2) U_j	:	$R_j \leftarrow$ generateRandom();
(3) U_j	:	$keyGenMsg_j \leftarrow$ '$SHARED\text{-}KEYGEN$' $\|\|NRN_j \|\| CardNb_j \|\| otherUserInfo$
(4) U_j	:	$Sig_j \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}_{\mathsf{M}}(keyGenMsg_j))$
(5) U_j	:	$E_{H_j} \leftarrow \mathsf{encrypt}(\mathsf{hash}_{\mathsf{S}}(Sig_j \|\| "F" \|\| R_j), proxyCert_o.\mathrm{pk})$
(6) $U_j \rightarrow U_o$	:	R_j, E_{H_j}
(7) U_o	:	$H_j \leftarrow \mathsf{decrypt}(E_{H_j}, PR_o)$
(8) U_o	:	$\Delta_j \leftarrow seed \oplus H_j$
(9) $U_o \rightarrow S$	:	appendKeyFile("$.F.keys$", [U_j, R_j, Δ_j])

Table 5.9: Extending the set of users with secure communication

5.4.4 Evaluation

Figure 5.4 gives an overview of the number of pin requests and cryptographic operations for each application. The former is relevant to evaluate the user-friendliness. The latter has

		eID		Proxy	Symm. Key	Hash
		PIN	Sig / Auth			
Local Secure Storage	1. *store* 2. *Store (no obf)*	1 1	1/0 1/0	no No	2 1	4 3
Shared Secure Storage	1. *setup* 2. *en-/decryption* 3. *extend group*	1 1 1	1/0 1/0 1/0	no no no	- - -	2 2 2
Deleg. to device	1. *setup*	1	1/0	yes	-	(2)
Deleg. to indiv.	1. *setup*	1	1/0	yes	-	(2)
Secure comm.	1. *setup*	-	-	yes	(2)	-
Secure email	1. *setup* 2. *receive contact* 3. *send/receive*	3 2	2/2 1/2	yes yes Yes	4 4	
Extended Shared Secure Storage	1. *extend group*	1	1/0	No	-	2

Figure 5.4: Application overview.

an impact on the performance. In many applications, a reasonable trade-off must be found between multiple non-functional qualities (f.i. performance, security/privacy, userfriendliness, degree of mobility, ...). Slight modifications to the protocols may increase/decrease the quality of some parameters. This is argued with some examples.

Proxy certificates can be used to access files in the shared secure storage application (instead of certificates on the eID card). This may increase the *mobility* and *user-friendliness* as individuals can now retrieve/update files on platforms without a smart card reader. Moreover, the user does not need to enter his pin code when accessing a file. However, the user must run the sharedKeyExtension again to generate a *Sig*' using his proxy certificate. This allows him to access the shared storage with a mobile device. However, the private key of the proxy certificate must be stored securely on the mobile device. Otherwise, the security properties decrease.

Index obfuscation can be omitted to increase the *performance* of the local secure storage application. If so, less cryptographic operations are required. However, this may have a negative impact on the *privacy* properties (i.e. an attacker can derive which individuals keep data at the local store). However, this is acceptable in certain settings (such as work stations in a home environment).

The extended shared secure store has better *security* properties compared to the shared secure store. The extension allows users to send the hash H_j over a secure channel. However, the proxy certificate (and corresponding private key) must be available.

The secure storage extension is more *user-friendly* if the authentication certificate on the eID card is used to generate the *Sig*(i.e. the pin code must only be entered once). However, it might be more feasible to use the signature certificate for *security* reasons.

5.5 Conclusion

The eID extensions allow to build more advanced applications based on the Belgian eID card. Together with the Identity Framework, these tools assist the software developer to build eID applications. A secure e-mail demonstrator was built to evaluate these tools.

Chapter 6

General conclusions and future directions

6.1 Summary of main contributions

This report described the lessons learnt from a two years research project that focuses on the development of eID applications in the commercial domain. It summarizes our experiences with application development for the Belgian eID technology and provides guidelines for the future development of advanced applications with the Belgian eID technology. The main contributions are classified according to four categories:

- The report gives a detailed overview of the *opportunities and constraints of the Belgian eID technology*. More specifically, a critical security and privacy assessment elaborates on the new threats that arise when introducing the eID technology in new domains (commercial domain, e-health domain, financial domain, ...). A prototypical attack in this category exploits the strategic decision to support SSO for authentication. It means that individuals only have to enter their pin code once to authenticate to multiple e-services. However, this implies that malicious applications can authenticate transparantly to other service providers on behalf of the user. Whereas this attack is less serious for governmental applications (i.e. governmental institutions already exchange a lot of information about individuals), this threat becomes very relevant for many commercial applications.

- Multiple *eID extensions* are presented. The extensions aim at (1) increasing the level of security and privacy, (2) facilitating access to personal information and (3) extending the security functionality of the Belgian eID technology. For instance, an advanced authentication protocol tackles certain security threats that arise when individuals authenticate with their eID over SSL. BeID proxy certificates is another example. They allow users to access personal information even at platforms where no card reader is available. Moreover, encryption keys can be kept in proxy certificates. Hence, individuals can set up secure communication channels or send confidential messages without a trusted third party. Finally, a mechanism is presented to generate symmetric keys based on the eID card. The software extensions might increase the application domains of the Belgian eID technology and/or the quality of existing applications (in terms of usability, mobility, security, privacy, ...).

- *A framework offers supports for the development of eID applications.* First, simple and intuitive APIs are offered to application developers. Second, it can support instantions of complex cryptographic building blocks (such as pseudonym certificates and anonymous credentials) while hiding the complexity to application developers. The framework aims at accelerating the development of advanced eID applications.
- Multiple applications are built to demonstrate the applicability of the eID extensions and the feasibility of the framework. Moreover, different variants are presented. It is clear that the security and privacy properties of an application depend on the concrete building blocks that are used. In many applications that were developed, the eID is used as bootstrap (i.e. at registration) to retrieve privacy-friendly tokens.

Although these contributions may support and accelerate the design of new qualitative applications with the current Belgian eID card, not all shortcomings can be tackled. The next section explores future research directions in eID technology.

6.2 Future directions in eID technology

This section explores directions for future eID technologies. A first solution takes privacy friendly identity files as the underlying technology for future eID cards. A second, more complex solution, is based on anonymous credentials. Finally, it is expected that eID technologies will shift from card specific solutions to *mobile identities* that are kept and managed on mobile devices. The latter are discussed in the last section.

6.2.1 Privacy Friendly Identity Files and Domains

With the current eID card, it is only possible to disclose the entire identity, address and picture files. Otherwise, the server cannot check if the signature of the NRB on the identity file is valid. This section introduces more privacy friendly identity files (PFID-files). The concept allows for releasing only the necessary personal attributes.
The PFID-file contains for each attribute the *hash* of its value. Hence, it is not possible to extract personal information out of the hash values without knowing the attribute value. To reduce brute-force or dictionary attacks, the hash function is randomized by adding an attribute specific random number to the plain text value: hash($ATTRIBUTE \parallel rand_{ATTRIBUTE}$). The PFID-file is certified (signed) by the National Registration Bureau (NRB). To disclose certified personal attributes, the card releases the signed PFID-file together with the plain text values and the attribute specific random numbers of these attributes: $ATTRIBUTE$, $rand_{ATTRIBUTE}$. The card should request the user's consent before releasing personal data. This consent could be given by entering a PIN code or pressing the OK button on a card reader with a separate PIN pad. The other party can verify the values by calculating the hashes and comparing them with the values in the PFID-file. Optionally, the user's consent could be overridden (no PIN code required) after proper authentication by privileged service providers. The latter can be useful for border control, police, emergency services, ...
Some attributes can be *encrypted* in the identity file. Instead of just storing the National Registry Number (NRN) as attribute value, the NRN can be encrypted with a symmetric key only known by the government or by another trusted third party (TTP). The enciphered NRN serves as a unique pseudonym and can - in case of abuse - be deanonymized by that TTP during a legal investigation.

Attribute values	PFID-file for *DOMAIN*
Nym_{DOMAIN}	$\text{encrypt}_{K_{NRB}}(DOMAIN \parallel \text{NRN})$
Name	$\text{hash}(\text{Name} \parallel rand_{(DOMAIN,Name)})$
Surname	$\text{hash}(\text{Surname} \parallel rand_{(DOMAIN,Surname)})$
Street	$\text{hash}(\text{Street} \parallel rand_{(DOMAIN,Street)})$
Zip code	$\text{hash}(\text{Zip code} \parallel rand_{(DOMAIN,ZipCode)})$
Municipality	$\text{hash}(\text{Municipality} \parallel rand_{(DOMAIN,Municipality)})$
Birth location	$\text{hash}(\text{Birth location} \parallel rand_{(DOMAIN,BirthLocation)})$
Birth date	$\text{hash}(\text{Birth date} \parallel rand_{(DOMAIN,BirthDate)})$
Hash photo	$\text{hash}(\text{Hash photo} \parallel rand_{(DOMAIN,Hashphoto)})$
...	...
	signature of the National Registration Bureau on the PFID-file for *DOMAIN*

Table 6.1: Overview of privacy friendly identity file

PFID-files on eID card. A PFID-file contains no personal information and does not need to be protected. However, in order to be sure that released personal attributes really belong to the card owner (and are not simply copied from another card), it is necessary to have that owner authenticate to the service provider. The service provider then needs to verify whether the PFID-file and the authentication certificate refer to the same holder and whether the certificate is still valid.
The attribute specific random number can be calculated from a master random number: $rand_{ATTRIBUTE} = \text{hash}(masterRandom \parallel ATTRIBUTE)$; and can be calculated at runtime. Hence, the card only needs to store (1) the plain text values of the personal attributes, (2) the master random number, (3) the signature of the NRB on the PFID-file and (4) the PIN-code for the user's consent. However, this requires that the card can calculate the hash at runtime. When no hash function is available, more storage is needed to store all the attribute hashes. To implement PFID-files, the API must be extended to pass the list of requested attributes to the card.

Multiple domains. With only one PFID-file, multiple actions of the same citizen can easily be linked. To reduce linkability, multiple PFID files can be created and signed by the NRB. Each file is assigned to a domain and should only be used in that domain: e.g. "GOVERNMENT", "COMMERCIAL", "MEDICAL", etc. Linkability is then only possible within one domain. A similar technique is used in the German eID card[19]. That card also works with domains.

To build multiple PFID-files, the hash values must be different in order to prevent linkability. A unique domain is concatenated to the master random value and the attribute name to calculate the random value (see table 6.1):

$rand_{(DOMAIN,\ ATTRIBUTE)} = hash(MasterRandom \| DOMAIN \| ATTRIBUTE)$

For encrypted attributes, the domain name precedes the actual attribute value. Hence, the cipher text is different for each domain. The cipher text of the domain and the NRN can be considered as a domain specific pseudonym for the card holder.

To link PFID-files to card owners, different authentication certificates are necessary each

referring to its own Nym_{DOMAIN}. Each certificate corresponds to its own keypair: ($SK_{auth_{DOMAIN}}$, $PK_{auth_{DOMAIN}}$). The only difference between the certificates is (1) the subject field, (2) the public key and hence (3) the signature of the certificate. The subject value is the domain pseudonym. Hence, each certificate can be linked to the corresponding PFID-file for each domain.
If only one PFID-file is used on the eID card, the additional storage space that is required compared to the current eID card is very small. An additional master random value needs to be stored.
When using multiple domains, more space is needed on the card. However, the extra space per domain is quite small. For each PFID-file, an extra Nym_{DOMAIN}, other encrypted attributes and the NRB's signature per PFID-file must be stored. Also, room for an extra authentication keypair and the signature on the domain specific authentication certificate must be provided.
Sometimes, the PFID-files need to be *updated* (e.g. if a citizen moves to a new address). This implies that the signatures of all PFID-files need to be updated. The NRB must build the new PFID-files by asking the master random value from the eID card. This is done after mutual authentication. This step is also needed with the current eID card to update the address file. As the NRN cannot be changed without replacing the eID card (because it is printed on it) the pseudonyms for the domains will be the same. Hence, the authentication certificates do not need to be updated.

6.2.2 Anonymous credentials

Anonymous credentials have been shown to be valuable technologies enabling anonymous, yet accountable transactions. They address the most important short-commings of the current eID technology with respect to privacy. Though, some proof-of-concept implementations exist, extended research for efficient anonymous credential systems is still required. For instance, anonymous credentials are currently too expensive with respect to processing power, touse them on smart cards. Hence, in the next few years it is not yet feasible to deploy a new eID card based on anonymous credentials. However, the current Belgian eID can be used as a bootstrap.

eID as a bootstrap

After eID authentication, a trusted party may issue an anonymous credential, containing certified information. This credential can then be used for services without the risk of exposing excessive identity information. Moreover, obtaining anonymous credentials is not limited to users with a Belgian eID. Other means of authentication may be used.
The ePoll application demonstrates the use of the Belgian eID as a bootstrap. The credential received after eID authentication can be used for signing electronic polls anonymously. Signatures are not linkable, yet it is not possible to sign a poll twice.
Additionally, anonymous credentials obtained using the Belgian eID can be integrated on mobile devices.
Future eID technology should use anonymous credentials. Selective disclosure allows to use the same credential for multiple domains. The user's privacy is preserved while it provides accountability for service providers. However, research is still needed to allow the use of anonymous credentials on resource constrained devices. Moreover, to make anonymous cre-

dentials really practical, new standards are critical.

6.2.3 Mobile identities

There is undoubtedly an increasing tendency towards mobile communications and mobile applications. It is expected that mobile devices will become the main guardians and managers of our multiple electronic identities for a broad range of applications and services which include payments, e-health, e-government etc.
Mobile devices will become the natural user interface in a ubiquitous computing environment, through which users will access services and perform their daily transactions. Most communication will be wireless and other parties can be malicious. The mobile should protect the interests of all stakeholders: (1) the user who wants to protect his privacy and prevent identity fraud or theft, but also wants to be able to use his rights to access preferably highly customized services; (2) the service provider that needs to verify the user's rights to access services, and should be able to get access to profiling information in order to customize the offered services; moreover –in case of abuse– it needs appropriate evidence to be able to hold the abuser accountable; (3) the authorities that wish to punish unethical behavior such as money laundering, computer crime, ...

Bibliography

[1] The ADAPID project website. https://www.cosic.esat.kuleuven.be/adapid/.

[2] Be-Health. http://www.behealth.be/.

[3] Belgian certificate revocation list. http://status.eid.belgium.be/.

[4] The Belgian Rijksregister website. http://www.ibz.rrn.fgov.be/.

[5] Duemmegi s.r.l. home and building automation http://www.duemeggi.it.

[6] Idemix. https://www.zurich.ibm.com/idemix.

[7] KBC: Pin-verification error - stop the disturbing active processes of the e-id card. http://www.kbc.be/welkom.

[8] KeyTrade. http://www.keytrade.be/.

[9] Kruispuntbank van de sociale zekerheid. http://www.ksz-bcss.fgov.be/.

[10] Microsoft Office. http://office.microsoft.com/.

[11] Mijn dossier. https://www.mijndossier.rrn.fgov.be/.

[12] My Belgium, e-government services. http://my.belgium.be.

[13] Nikobus - home automation http://www.niko.eu.

[14] Open Office. http://www.openoffice.org/.

[15] SafeBoot. http://www.safeboot.com/.

[16] Tax-on-web. http://www.taxonweb.be/.

[17] Tele Ticket Service. http://www.teleticketservice.com/.

[18] U-prove. http://www.credentica.com/.

[19] Advanced security mechanisms for machine readable travel documents extended access control (eac) and password authenticated connection establishment (pace). Technical Guideline TR-03110, 2008.

[20] Patrick Andries. *eID Middleware Architecture Document*. Zetes, 1.0 edition, 2003.

[21] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 78–91, New York, NY, USA, 1997. ACM.

[22] S. Brands. A technical overview of digital credentials, 1999.

[23] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 93–118, London, UK, 2001. Springer-Verlag.

[24] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[25] Bart De Decker, Vincent Naessens, Jorn Lapon, and Pieter Verhaeghe. Kritische beoordeling van het gebruik van de belgische eid kaart. CW Reports CW524, Department of Computer Science, K.U.Leuven, May 2008.

[26] Antonio J. de Vicente, Juan R. Velasco, Ivn Mars-Maestre, and Alvaro Paricio. A proposal for a hardware architecture for ubiquitous computing in smart home environments. In *Proceedings of the I International Conference on Ubiquitous Computing: Applications, Technology and Social Issues*, 2006.

[27] Armando Roy Delgado, Rich Picking, and Vic Grout. On context in authorization policy. In *Proceedings of the 6th International Network Conference (INC 2006)*, pages 357–366, 2006.

[28] Lander Dufour. Integratie van e-ID technologie in domotica software. Master's thesis, KaHo Sint-Lieven, Gent, work in progress.

[29] Yi-Min Wang et al. Towards dependable home networking: An experience report, 2000.

[30] Sebastien Gamby, Laurent Schumacher, and Jean Ramaekers. Securisation of SIP Presence notifications thanks to the Belgian electronic identity card. *ngmast*, pages 125–129, 2007.

[31] ENISA Giles Hogben. ENISA Position Paper No.1: Security Issues and Recommendations for Online Social Networks. 2007.

[32] Wolfgang Granzer, Wolfgang Kastner, Georg Neugschwandtner, and Fritz Praus. Security in networked building automation systems. *Factory Communication Systems, 2006 IEEE International Workshop on*, pages 283–292, 2006.

[33] Matt Hooks and Jadrian Miles. Onion routing and online anonymity. *CS182S*, 2006.

[34] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News*, 32(1):60–65, 2001.

[35] Michiel Ide, Tom Deryckere, and Luc Martens. Exploiting the Benefits of an Electronic Identity Card in an Interactive Television Environment. *ccnc*, 0:809–810, 2008.

[36] V. Kapsalis, K. Charatsis, A. Kalogeras, and G. Papadopoulos. Web gateway: A platform for industry services over internet, 2002.

[37] Cory D. Kidd, Robert Orr, Gregory D. Abowd, Christopher G. Atkeson, Irfan A. Essa, Blair MacIntyre, Elizabeth D. Mynatt, Thad Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *Cooperative Buildings*, pages 191–198, 1999.

[38] L-Sec. Secure use of the eID. 2006.

[39] Jorn Lapon, Koen Vangheluwe, Vincent Naessens, and Annemie Vorstermans. A Generic Architecture for Secure Home Automation Servers. In *Proceedings of the Third European Conference on the Use of Modern Information and Communication Technologies*, pages 261–271, Gent, 2008.

[40] Jorn Lapon, Bram Verdegem, Pieter Verhaeghe, Vincent Naessens, and Bart De Decker. Extending the Belgian eID technology with mobile security functionality. In *Lecture Notes of the International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec)*, volume 17, pages 97–109. Springer, June 2009.

[41] Jorn Lapon, Kristof Verslype, Pieter Verhaeghe, Bart De Decker, and Vincent Naessens. PetAnon: a fair and privacy-preserving petition system. In *The Future of Identity in the Information Society. Challenges for Privacy and Security: Pre-proceedings*, pages 73–78. FIDIS/IFIP Internet Security & Privacy Summer School, September 2008.

[42] Emil C. Lupu and Morris Sloman. Towards a role based framework for distributed systems management. *Journal of Networks and Systemss Management, Plenum Press*, 1997.

[43] A.Malpani S. Galperin C. Adams M. Meyers, R. Ankney. Rfc2560: X.509 internet public key infrastructure, online certificate status protocol - ocsp. `http://tools.ietf.org/html/rfc2560`, June 1999.

[44] P. McDaniel. On context in authorization policy. In *In Proceedings of the ACM Symposium on Access Control Models and Technologies*, pages 80–89, 2003.

[45] Zhang Lee Ni. Open service residential gateway for smart homes.

[46] D. M. Goldschlag P. F. Syverson and M. G. Reed. Anonymous connections and onion routing. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 1997. IEEE Computer Society.

[47] W. Ford D.Solo R. Housley, W. Polk. Rfc3280: Internet x.509 public key infrastructure, certificate and certificate revocation list (crl) profile. `http://tools.ietf.org/html/rfc3280`, April 2002.

[48] W. Polk D.Solo R. Housley, W. Ford. Rfc2459: Internet x.509 public key infrastructure, certificate and crl profile (obsolete). `http://tools.ietf.org/html/rfc2459`, January 1999.

[49] Guy Ramlot. *eID Hierarchy and Certificate Profiles*. Zetes, Certipost, 3.1 edition, 2006.

[50] Anthony Rhodes and William Caelli. A review paper: Role based access control. 1999.

[51] Johan Rommelaere. *Belgian Electronic Identity Card Middleware Programmers Guide.* Zetes, 1.40 edition, 2003.

[52] R. Sabett C. Merrill S. Wu S. Chokhani, W. Ford. Rfc3647: Internet x.509 public key infrastructure, certificate policy and certification practices framework. `http://tools.ietf.org/html/rfc3647`, November 2003.

[53] José Sahún. Working committe 6: Utilization of gases for domestic, commercial and transportation sector – study group 6.1: Gas appliances for the 21st century. part 3 – services., 2003.

[54] Umar Saif and David J.Greaves. Communication primitives for ubiquitous systems or rpc considered harmful. In *ICDCSW '01: Proceedings of the 21st International Conference on Distributed Computing Systems*, page 240, Washington, DC, USA, 2001. IEEE Computer Society.

[55] Zigor Salvador, Raúl Jimeno, Alberto Lafuente, Mikel Larrea, and Julio Abascal. Architectures for ubiquitous environments, 2004.

[56] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[57] Marc Stern. *Belgian Electronic Identity Card content.* Zetes, CSC, 2.2 edition, 2003.

[58] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Rfc 3820 - internet x.509 public key infrastructure (pki) proxy certificate profile., 2004.

[59] Martijn H. Vastenburg and David V. Keyson. Creating and modeling the user experience in a residential gateway environment.

[60] Pieter Verhaeghe, Jorn Lapon, Bart De Decker, Vincent Naessens, and Kristof Verslype. Security and privacy improvements for the Belgian eID technology. In *Emerging Challenges for Security, Privacy and Trust*, volume 297, pages 237–247. Springer, May 2009.

[61] Pieter Verhaeghe, Jorn Lapon, Vincent Naessens, Bart De Decker, Kristof Verslype, and Girma Nigusse. Security and Privacy Threats of the Belgian Electronic Identity Card and Middleware. In *EEMA European e-Identity conference*, Den Haag, June 2008.

[62] Kristof Verslype, Bart De Decker, Vincent Naessens, Girma Enideg Nigusse, Jorn Lapon, and Pieter Verhaeghe. A privacy-preserving ticketing system. In *Lecture notes in computer science*, volume 5094, pages 97–112. Springer, July 2008.